МОНИТОР

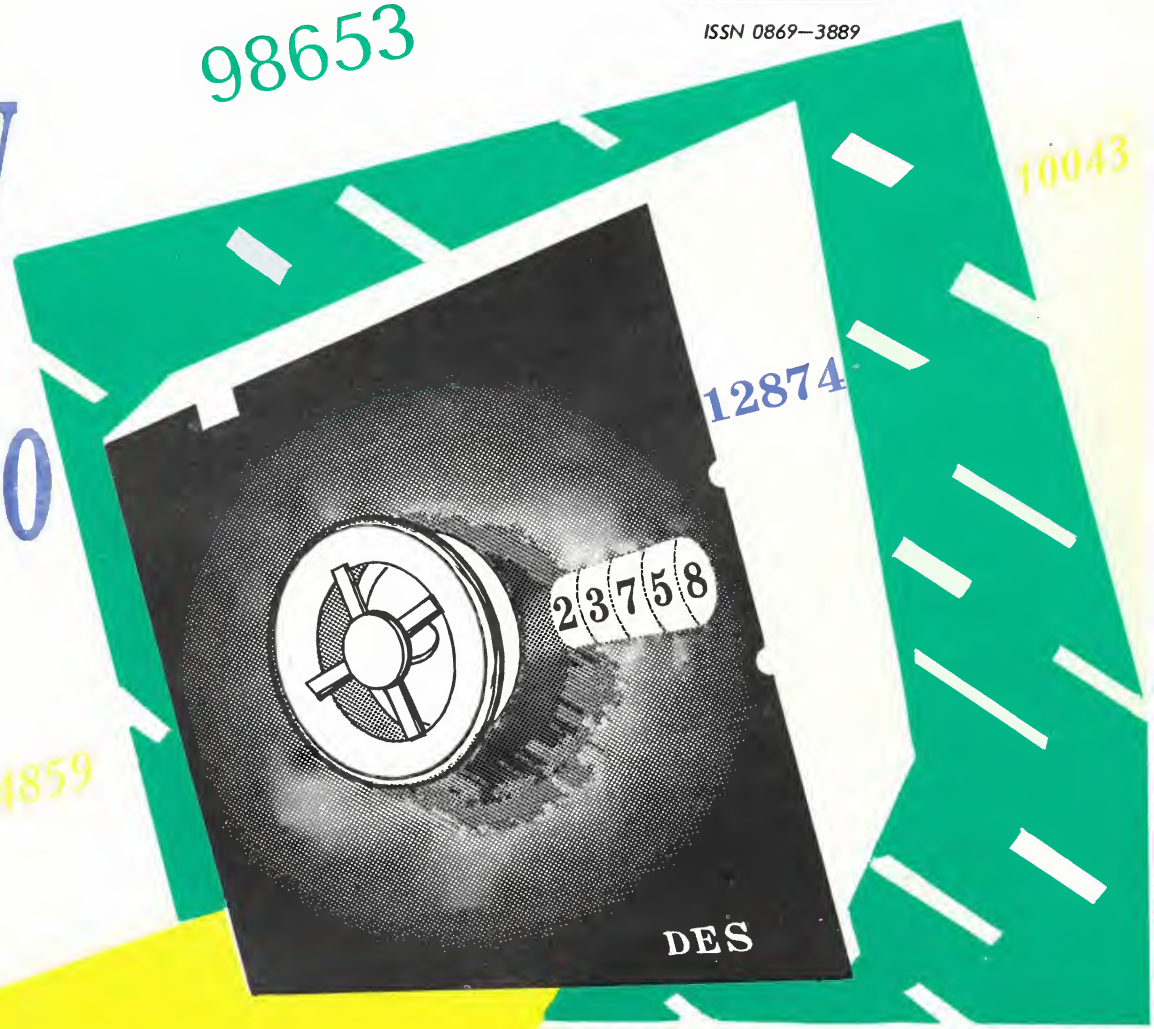
журнал для программистов — практиков

6–7.92

ISSN 0869–3889

Тайну знает только DES

стр. 14



PROGRAMMER

Мирное применение вирусной технологии

DOS и BIOS–идем тропой Хо Ши Мина

Сетевой протокол IPX

USER

От зубила до звездолета: история на экране

AUTODESK®

МЫ СТРОИМ НАШ ОБЩИЙ ДОМ

Продукция компании Autodesk:

AutoCAD R12

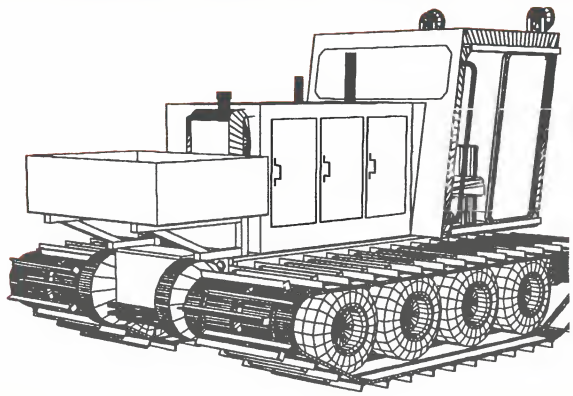
AutoCAD R11

Автокад вер. 11 386

Автокад вер. 11 286

РОК

AME



AutoLISP Compiler

HyperChem R2

Autodesk Animator Pro

Autodesk 3D Studio R2

AutoSketch вер. 3

AutoShade/AutoFlix

за рубли и СКВ

ПРОДУКЦИЮ КОМПАНИИ AUTODESK ПОДДЕРЖИВАЮТ БОЛЕЕ 100 ОФИЦИАЛЬНЫХ ДИЛЕРОВ, 30 УЧЕБНЫХ ЦЕНТРОВ, 60 РАЗРАБОТЧИКОВ ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, СРЕДИ КОТОРЫХ:

АП-Технология
664005 Иркутск Профсоюзная 10а
тел. 3952-244894

Бюро ESG
197022 С.-Петербург а/я 119
тел. 812-2529490

Интелграф
103001 Москва а/я 119
тел. 095-2915716
Гарантия качества услуг в области информационных технологий

Интехцентр
109028 Москва Астаховский 1/2
тел. 095-2270616
АСУ гостиниц и банков, локальные сети, телефонизация офисов

Ками
443090 Самара Антонова-Овсеенко 44
тел. 8462-222006
Вы+Мы+Автокад=Решение всех проблем

Когра
226001 Рига Сколас 21
тел. 0132-273052
Обучение программным продуктам 3D Studio и Animator

Куб
113114 Москва 4 Кожевнический 2/12
тел. 095-2357060
AutoЕСКД, AutoЕСПД - безблочная технология, коммерческая информация

Магнон
173003 Новгород Ленинградская 41
тел. 81600-29102

Мета
252039 Киев 40 лет Октября 50
тел. 044-2639042
Предлагаем систему архитектурного проектирования МАЭСТРО

Найт
630081 Новосибирск Д.Бедного 49
тел. 3832-664753

Российская литейная компания
123363 Москва Яна Райниса 1
тел. 095-9498445

Селена
400006 Волгоград-6 Ленина 225/65
тел. 8442-713688
Программно-технические комплексы, сети, вычислительная техника

SPEA-SPAGA
117296 Москва Ломоносовский 16
тел. 095-1332010
Графические платы высокого разрешения и мониторы, САПР печатных плат

Стиплер
119034 Москва Кропоткинская, 40
тел. 095-2462499

Тема
115487 Москва Садовники 4
тел. 095-1186356

Унирем
197198 С.-Петербург Добролюбова 6/2
тел. 812-2352398

Урал-Диалог
454080 Челябинск Ленина 79
тел. 3512-336644
ПО, компьютеры, оргтехника - продажа и обслуживание

УралКАД
454080 Челябинск а/я 8298
тел. 3512-399837
Поставка и разработка программной продукции, обучение

Уралмаш-Интел
620012 Екатеринбург
Машиностроителей 12
тел. 3432-374976

ЭКОММ
252195 Киев Л.Украинки 26
тел. 044-2949086
Интегральные решения САПР технологий

С 1 СЕНТЯБРЯ ПО 31 ДЕКАБРЯ МЫ ПРОДАЕМ ЛИЦЕНЗИИ НА ИСПОЛЬЗОВАНИЕ НЕЛЕГАЛЬНЫХ КОПИЙ АВТОКАДА. У НАС ВЫ МОЖЕТЕ КУПИТЬ ПОЛНЫЙ КОМПЛЕКТ ДОКУМЕНТАЦИИ ПО АВТОКАДУ ВЕР. 10

Autodesk RF, Россия, 107005, Москва, 2 Бауманская ул., 9/23
тел. (095) 261 63 63
факс (095) 267 99 27

ПРИГЛАШАЕМ ВАС ПОСЕТИТЬ НАШ СТЕНД НА ВЫСТАВКЕ "КОНВЕРСИЯ - 92" (СОКОЛЬНИКИ, 7 - 14 ДЕКАБРЯ 1992 ГОДА)

Одна из важных целей, которую мы преследуем при издании журнала "Монитор", - содействие отечественным программистам в разработке и распространении программных продуктов. Поскольку читатели нашего журнала - программисты, мы обращаем особое внимание на разработчиков инструментальных средств.

В настоящее время прошли конкурсный отбор и приняты к распространению через журнал следующие программные продукты, разработанные авторами нашего журнала.

1. Турбо Гамес - технология создания графических приложений и полный набор инструментов для ее реализации. Турбо Гамес выполнен в виде библиотеки для Турбо Паскаль 6.0. Он позволяет работать с графикой и звуком на профессиональном уровне. Отличное средство для создания компьютерных игр (и не только их). Разработчик - Д.Солдатенков (статьи в номерах 4.92, 5.92, 6-7.92).

2. HyperRef - технология и набор инструментов для создания гипертекстовых систем. С помощью HyperRef Вы можете создавать, модифицировать и сопровождать гипертекстовые документы любого объема (книга, словарь, энциклопедия, справочник), включающие текст, иллюстрации, программы. Разработчики - Д.Гришечкин, В.Оленев, А.Тихонов (статьи в номерах 1.91, 2.91, 3.92, 5.92).

3. АвтоГРАФ - пакет для создания на экране дисплея графиков с результатами обработки числовых данных и последующей высококачественной печати на 9-игольчатых и 24-игольчатых принтерах. АвтоГРАФ предоставляет самые разнообразные возможности построения графиков в соответствии со стандартами ЕСКД. Разработчик - Н.Косарев (статьи в номере 4.92 и в текущем номере).

Те, кто заинтересован в приобретении этих программных продуктов (а к моменту выхода данного номера их список, вероятно, расширится) могут обращаться к нам в редакцию (тел. (095) 238-99-33, факс (095) 238-32-78). Мы приглашаем к сотрудничеству в распространении журнала и программных продуктов частных лиц и организации из разных регионов страны.

Если Вы разработали оригинальный высококачественный программный продукт, но испытываете сложности с его рекламой и распространением, приходите к нам, возможно, мы сумеем Вам помочь.

Дерзайте! Желаем Вам успеха. Audaces fortuna iuvat.

Журнал для программистов-практиков

Редколлегия:
Д.Э.Бройтман
(главный редактор)
А.А.Винокуров
(отдел распространения)
В.В.Водолазский
Е.В.Гун
(отдел подписки)
И.Г.Гун
С.В.Макельский
(главный художник)
А.М.Миляков
(ответственный секретарь)
Н.А.Никитина
(отдел рекламы)
А.В.Орехов
(издательский директор)
А.В.Семенников
(компьютерная верстка)
Л.И.Турок
Л.И.Устинова

Представительство в Европе (Германия):
тел. 8-(0631)-7-49-05,
E-mail: kphy76@klio.rhrk.uni-kl.de
Марина Гетманская

В оформлении номера использованы рисунки
К.З.Гусалова
и А.С.Замулы

Подписано в печать с оригинал-макета 03.11.92.
Формат 60x84/8.
Тираж 30 000 экз.
Печать офсетная. Печ.л.8.0.
Изд. N1971.
Цена договорная
Зак. 2093
Ф-ка "Картолитография"

Издатель:
Научно-техническое общество

Софт-Москва

Для писем: 117419, г.Москва,а/я 765
Тел. (095)238-99-33
Факс (095)238-32-78

© "Монитор", 1992

Содержание

Технологии программирования

Д.Солдатенков. **Спрайты. Путь от графического редактора к компьютерным играм. Часть 2** — 5

Демонстрируется технология создания и вывода на экран спрайтов — небольших объектов, постоянно меняющих свою форму и положение.

Защита программ

И.Василенко, Е.Гулаков. **Защищайтесь, сударь! Часть 3** — 7

Из третьей части статьи читатели узнают, как сделать программу "привязанной" к "своему" компьютеру, как ограничить возможность использования программ путем задания счетчика запусков или учета времени, в течение которого они могут работать.

В.Водолазкий. **Коммерческие системы шифрования: основные алгоритмы и их реализация. Часть 1** — 14

Первая из серии статей, посвященных алгоритмам шифрования коммерческой информации. В статье детально описан алгоритм DES.

О.Сурин. **Мирное применение вирусной технологии** — 21

Если Вы умеете создавать вирусы, то Ваши знания могут быть направлены в полезное русло. Защита программ от запуска с помощью вируса - тема предлагаемой статьи.

Компьютер в разрезе

В.Водолазкий **Скрытая мощь старенького EPSON** — 19

Статья посвящена настройке принтеров фирмы EPSON серий FX,LX,MX. Предлагаемая программа позволяет печатать на одном стандартном листе бумаги до 120 строк.

Н.Косарев. **Копия экрана? Это очень просто!** — 28

Как напечатать копию экрана на матричном принтере типа EPSON в 8-игольном графическом режиме.

Игры

Постижение истории — 31

Экспертный анализ популярной игровой программы Civilization фирмы Microprose.

Содержание

Базы данных

С.Крапивенко. **Clipper или Paradox: попытка аргументированного выбора Часть 3** 35

Третья часть статьи, сравнивающей возможности Clipper и Paradox.

DOS под микроскопом

В.Рабец. **Работа с DOS и BIOS в обход прерываний** 41

Статья рассказывает о методах поиска точки входа в обработчики прерываний DOS и BIOS и способах обращения непосредственно к этим обработчикам. Приведен модуль OriginOS, реализующий данные методы на Turbo Pascal 6.0. Статья содержит некоторые сведения о языке ассемблера, достаточные для понимания работы ассемблерных процедур OriginOS.

Сети

А.Ермолов. **Что такое IPX и как его использовать в своих целях Часть 1** 48

В статье приводится краткое описание программы симметричного обмена информацией – IPX, входящей в состав сетевых операционных систем семейства NetWare фирмы Novell. Приводятся простейшие примеры применения программного интерфейса IPX в пользовательских программах.

Этюды для программистов

С.Курковский. **Алгоритмы сортировки** 54

Проблема упорядочивания данных с практической точки зрения: достоинства и недостатки пяти различных методов сортировки.

Языки программирования

К.Злобин. **Aho+Weinberger+Kernighan=AWK** 59

В статье рассказывается о языке обработки текстов AWK.

М.Черкашин. **Работаем с таблицами** 60

Третья статья из цикла, посвященного проектированию компиляторов. Рассказывает о том, как реализовать в компиляторе схему синтаксически управляемого перевода.

Бизнес

"У CA самые серьезные намерения в России..." 63

Интервью с президентом представительства фирмы Computer Associates в России Александром Гольбергом

Всем, кто верит в компьютерные технологии

ТОО "РАКУРС"

Официальный дистрибьютер
НПЦ САПР САПСАН
ПРЕДЛАГАЕТ рабочую станцию *БЕСТА*

Программное обеспечение—для любых классов задач :

- инструментальные средства, СУБД,
- X-Windows, OSF Motif,
- информацинно-поисковые системы:
  банковская деятельность,
  АСУ "Производство",
  АСУ "Больница",
  АСУ "Пожарная охрана" и т.д.
- графические и технологические системы:
  прочностный анализ,
  моделирование поверхностей,
  АРМ инженера-конструктора и т.д.

Современный уровень вычислительной техники:

- процессор Motorola 68020/68030,
- быстродействие – до 9 MIPS,
- тактовая частота 16/25/33 МГц,
- оперативная память 4/8/16/32/64 Мб,
- винчестерский накопитель до 1.5 Гб,
- стриммерный накопитель 155 Мб,
- графический контроллер 1280x1024, 256 цветов,
- сетевой контроллер Ethernet,
- наработка на отказ – не менее 30 000 часов.

Гарантия качества и надежности рабочей станции БЕСТА:

- современная технология :
  контроль элементной базы,
  автоматизированная сборка,
  пайка волной,
  тестирование модулей,
- импортные комплектующие и периферия,
- модульная компоновка "под заказчика".

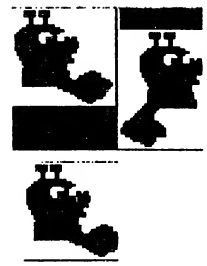
Рабочая станция БЕСТА—альтернатива сетям на базе ПК. Она обеспечивает одновременную работу 50 пользователей.

ТОО "РАКУРС" предлагает систему автоматизированного проектирования РАКУРС:

- пространственное проектирование поверхностей сложной формы и получение цветных реалистических изображений,
- подготовка программ для станков с ЧПУ для 3-х и 5-ти координатной обработки и объемное моделирование работы станка,
- генератор постпроцессоров для различных станков с ЧПУ.

Тел.: (095) 277-88-72

Спрайты. Путь от графического редактора к компьютерным играм



Часть 2

Д.Солдатенков

В предыдущей статье мы договорились, что будем рассматривать технологию изготовления спрайтов как состоящую из двух частей. Первая часть должна подготовить данные, "прямыми сделать стези" для быстрой процедуры вывода. Вторая часть должна как можно быстрее рисовать спрайт на экране. Способ вывода определяет формат данных для него, поэтому мы сначала рассмотрели вторую часть.

Была приведена программа на языке Турбо Паскаль версии 6.0 с использованием встроенного ассемблера, которая выводит спрайт в 16-цветном EGA режиме. Теперь посмотрим, как можно получать изображения спрайтов для этой программы при помощи графического редактора.

Прежде всего необходимо вывести изображение, созданное в графическом редакторе, на экран. Для этого можно воспользоваться одним из стандартных форматов. Например, формат PCX уже многократно описан и достаточно прост. Он поддерживается большинством редакторов. Существует множество библиотек для работы с этим форматом. Полный исходный текст одной из них был опубликован в журнале "Монитор", N3, 1992.

Далее для того чтобы воспользоваться приводимой ниже программой GetSprite (листинг), которая переведет спрайт в удобный для вывода формат, изображение должно быть специальным образом подготовлено в графическом редакторе. Все редакторы имеют функции, позволяющие легко провести такую подготовку, и при небольшом навыке она займет несколько секунд.

Изображение должно располагаться особым образом. Предположим, мы хотим создать человечка, машущего рукой. Две фазы движений человечка показаны на рис. 1.

Рис. 1.

После того как они нарисованы, каждую из фаз надо окружить белой рамкой, как показано на рис. 2.

Рис. 2.

Затем с каждой картинки надо сделать копию и расположить ее таким образом, чтобы верхняя белая кромка копии совпала с нижней белой кромкой оригинала, как показано на рис. 3.

Теперь те области, которые должны быть прозрачными, в нижних рамках следует

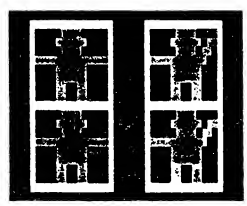


Рис. 3.

перекрасить в белый цвет, а в верхних они должны остаться черными. Эту операцию можно сделать при помощи заливки, как показано на рис. 4.

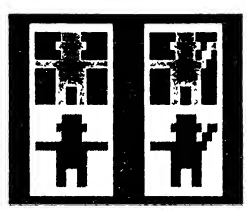


Рис. 4.

Если заливкой по какой-либо причине воспользоваться нельзя, можно сделать нижний прямоугольник полностью белым. Это не повлияет на работу про-

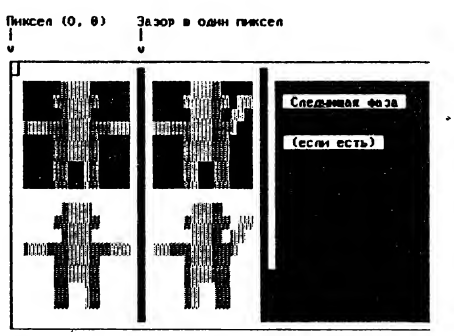


Рис. 5.

граммы перевода, если не считать того, что все черные пикселы станут прозрачными.

Запакованные таким образом фазы надо расположить в левом верхнем углу экрана так, как показано на рис. 5.

Полученный экран надо сохранить в PCX формате, выйти из графического редактора и запустить программу GETSPRT с двумя параметрами. Первый параметр - имя PCX файла, второй - имя файла, в котором будут храниться созданные спрайты. Предположим, мы записали нашу картинку в файл MAN.PCX и передали его как первый параметр, а в качестве второго параметра написали MAN:

```
GETSPRT MAN.PCX MAN
```

Программа создаст два файла: MSP.$A и MAN.$B. В каждом из этих файлов будет храниться по одной фазе и каждый из них можно использовать как спрайт, содержащий одну фазу, но для удобства пользования файлы можно склеить в один при помощи обычной системной команды COPY /B MAN.$A+MAN.$B MAN.SPR. Фазы при этом могут располагаться как угодно, но при загрузке в память они будут считаны и пронумерованы в том порядке, в котором они склеивались.

Для того чтобы программа вывода EGASprite, описанная в журнале "Монитор", N 5, 1992, обращалась за данными в файл, надо заменить процедуру MakeSprite (пример 1).

Пример 1.

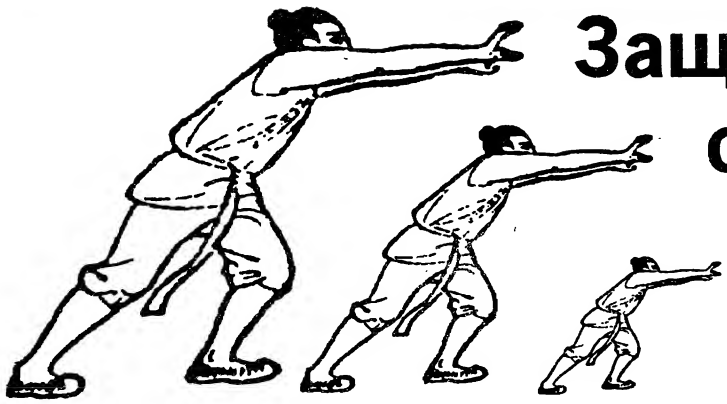
```
procedure MakeSprite;
var
  SprFile: File;
  Size: Word;
begin
  Assign (SprFile, 'MAN.SPR');
  ReSet (SprFile, 1);
  BlockRead (SprFile, Size , 2);
  BlockRead (SprFile, Sprite , Size-2);
  Close (SprFile);
end;
```

Теперь на экране будет перемещаться изображение человечка, которое было создано в графическом редакторе. Во время рисования необходимо следить за тем, чтобы размеры изображения были не больше, чем у одноцветного человечка, нарисованного в первоначальном варианте программы EGASprite. Если же хочется нарисовать бо́льшой спрайт, то надо изменить константы, ответственные за длину и высоту спрайта (SprLen и SprWid), соответствующим образом.

За несколько лет использования данной технологии была накоплена довольно обширная библиотека спрайтов. Если принять описанный формат для спрайтов, выводимых в 16-цветном EGA режиме, как стандартный, то может появиться возможность обмена. ■

Листинг

```
program GetSprt;
uses Graph, Crt,
     PCXPic;  { Полный исходный текст этой библиотеки }
              { приведен в журнале "Монитор", N 3, 1992 }
const
  GD: Integer = 3;          { Графический режим 640x350, 16 цветов }
  GM: Integer = 1;
  X1: Integer = 0;          { Координаты текущей фазы}
  Y1: Integer = 0;
  X2: Integer = 0;
  Y2: Integer = 0;
type
     PInteger = ^Integer;
var
  SprFile: File;            { Файл для спрайта}
  SprFileName, PCXFileName: String;    { Имена файлов спрайта и картинки}
  SprLen, SprWid: Byte;     { Размеры спрайта }
  Lth,                      { Длина файла спрайта}
  g: Integer;               { Результат показа картинки }
  IniBuff, Buff: ^Byte;     { Указатели на буфер}
  LBuff: LongInt absolute Buff; { Целое совмещено с указателем}
  HiPic, LoPic, BitMask: array [0..7] of Byte;
                            { Цвета на верхней и нижней}
                            { картинках, битовая маска}
function DetectNextSpr: Boolean;    { Определяет наличие }
var                                 {следующей фазы у спрайта, }
  X, Y: Integer;                    { выставляет размеры фазы}
begin
  X := X1; Y := Y1;
  while GetPixel (X, Y1) = 15 do Inc (X);
  X2 := X-1;
  while GetPixel (X1, Y) = 15 do Inc (Y);
  Y2 := Y-1;
  If (X2 < = X1) or (Y2 < = Y1) then DetectNextSpr := FALSE
  else DetectNextSpr := TRUE;
end;
procedure InitNextSpr;              { Начинаем переводить новую }
begin                               { фазу}
  SprLen := ((X2 - X1) - 1) div 8;  { Вычисляем длину спрайта }
  if (((X2 - X1) - 1) mod 8 < > 0) then Inc (SprLen);
  SprWid := (Y2 div 2) - 1;         { Вычисляем ширину спрайта}
  Lth := 5 * SprLen * SprWid + 4;   { Вычисляем длину файла }
                                    { спрайта}
  GetMem (Buff, Lth);               { Аллокируем буфер}
  IniBuff := Buff;                  { Запоминаем его начальный адрес }
  PInteger (Buff)^ := Lth; Inc (LBuff, 2); { Заполняем начало }
                                    { буфера}
  Buff^ := SprLen; Inc (LBuff);
  Buff^ := SprWid; Inc (LBuff);
  Assign (SprFile, SprFileName);    { Открываем новый файл для }
  ReWrite (SprFile, 1);             { хранения спрайта }
end;
procedure WriteResult;              { Запись в буфер через каждые 8 }
var                                 { точек экрана }
  I, J: Integer;
begin
  Buff^ := 0;
  for I := 0 to 7 do Buff^ := Buff^ or (BitMask[I] shl I); { Запись маски }
  Inc (LBuff);
  for J := 0 to 3 do begin          { Запись 4 байтов по плоскостям }
      Buff^ := 0;
      for I := 0 to 7 do begin
          BitMask[I] := 1 and (HiPic[I] shr J);
          Buff^ := Buff^ or (BitMask[I] shl I);
      end;
      Inc (LBuff);
  end;
end;
procedure WriteSprFile;             { Запись заполненного буфера в }
begin                               { файл и освобождение памяти }
  BlockWrite (SprFile, IniBuff^, Lth);
  Close (SprFile);
  FreeMem (IniBuff, Lth);
end;
procedure TakeSprite;       { Читаем информацию с графического экрана }
var
  I, J, K,
  X8Poi, XCur,              { X до начала байта и текущий }
  YHiPic, YLoPic: Integer;  { Y верхней и нижней картинок }
begin
  for J := 1 to SprWid do begin             { Цикл по Y }
      YHiPic := J;
      YLoPic := Trunc (Y2 / 2 + J);
      for I := 0 to SprLen-1 do begin       { Цикл по X }
          X8Poi := X1 + 1 + 8*I;
          for K := 0 to 7 do begin
              XCur := X8Poi+7-K;
              HiPic[K] := 0;
              if XCur > = X2 then BitMask[K] := 0    { Вышли за границу }
              else begin
                  HiPic[K] := GetPixel (XCur, YHiPic);
                  LoPic[K] := GetPixel (XCur, YLoPic);
                  if (HiPic[K] = 0) and (LoPic[K] = 15) then BitMask[K] := 0
                  else begin                { Прозрачность }
                      BitMask[K] := 1;
                      PutPixel (XCur, YHiPic, 14);
                      PutPixel (XCur, YLoPic, 14);
                  end;
              end;
          end;
          WriteResult;          { Записываем каждые 8 точек }
      end;
  end;
  WriteSprFile;                 { Запись очередного спрайта }
  X1 := X2+2;                   { Конец становится началом }
  Inc (Byte (SprFileName [Length (SprFileName)] )); { Другое расширение }
end;
begin
  InitGraph (GD, GM, '');
  PCXFileName := ParamStr(1);
  SprFileName := ParamStr(2) + '.$A';
  { Вывод на экран картинки в PCX формате. }
  { Полный исходный текст в журнале "Монитор", N 3, 1992. }
  g := ShowPCXFromFile (0, 0, GD, TRUE, NormalPut, PCXFileName);
  while DetectNextSpr do begin
      InitNextSpr;
      TakeSprite;
  end;
  CloseGraph;
end.
```



Защищайтесь, сударь !

Часть 3

И.Василенко, Е.Гулаков

Как Вы уже возможно заметили, "привязка" программы к дистрибутивной дискете путем описанных ранее способов защиты несет в себе большое неудобство для пользователя, связанное с необходимостью работы только со вставленной в дисковод оригинальной дискетой. Гораздо удобнее иметь необходимый программный продукт записанным на винчестере. Поэтому необходимо, чтобы контролирующая часть защищаемой программы (КЧЗП) "запомнила" свой компьютер и потом при запуске сравнивала имеющиеся характеристики с характеристиками "родного" компьютера. В случае их расхождения можно считать, что программа незаконно скопирована, и прервать ее выполнение. Для этого надо найти какие-то параметры, которые бы индивидуально характеризовали каждую вычислительную систему. На самом деле это весьма нетривиальная задача, поскольку открытая архитектура построения компьютеров IBM PC подразумевает их обезличенность.

Рассмотрим, что все же можно предложить для КЧЗП в качестве характеристик, которые могли бы проверяться при работе защищаемой программы.

Физические дефекты винчестера

При работе жесткого диска (винчестера) возможно возникновение сбойных участков на магнитной поверхности. Такие дефектные места могут иметься даже на совершенно новом винчестере. Номера этих сбойных участков помещаются в FAT (их признак - код FF7). При инсталляции защищаемой программы на винчестер в ее контролирующую часть записываются их адреса. В процессе выполнения программы осуществляется сравнение адресов сбойных участков, записанных в КЧЗП и в FAT. В случае запуска незаконно скопированной программы будет обнаружено расхождение сравниваемых адресов (первые не будут составлять подмножество вторых) и произойдет выполнение аварийных действий, предусмотренных для такого случая.

Развитием этой идеи является метод, в котором некоторые исправные кластеры помечаются как сбойные, и в них помещается информация для КЧЗП. (При копировании такие кластеры не передаются.)

Дата создания BIOS

Можно попытаться сузить класс компьютеров, на которых возможно функционирование незаконно скопированной программы. Это достигается, например, путем введения проверки даты создания BIOS, которая записана в ПЗУ каждого компьютера. В табл. 1 представлены возможные модификации и версии BIOS, записанной в памя-

Таблица 1. Модификации, версии BIOS и модели систем

Байт идентификации	Байт подмодели	Система	Дата	Ревизия
FF	00	PC	04/24/81	-
FF	00	PC	10/19/81	-
FF	00	PC	10/27/82	-
FE	00	PC XT	11/08/82	-
FD	00	PCjr	06/01/83	-
FC	00	PC AT	01/10/84	-
FC	00	PC AT	01/10/85	Rev. 1
FC	01	PC AT	11/15/85	Rev. 0
FC	02	PC XT 286	04/21/86	Rev. 0
FC	04	PS/2 Model 50	02/13/87	Rev. 0
FC	04	PS/2 Model 50Z	04/18/88	Rev. 3
FC	05	PS/2 Model 60	02/13/87	Rev. 0
FC	01	PC AT	01/13/88	Rev. 0
FB	00	PC XT	01/10/86	Rev. 1
FB	00	PC XT	05/09/86	Rev. 2
FA	00	PS/2 Model 30	09/02/86	Rev. 0
FA	00	PS/2 Model 30	12/12/86	Rev. 1
F9	00	PC-совместимые	09/13/85	Rev. 1
F8	00	PS/2 Model 80/16	03/30/87	Rev. 0
F8	01	PS/2 Model 80/20	10/07/87	Rev. 0

ти компьютера. Эта же дата заносится в КЧЗП, и в процессе выполнения защищаемой программы осуществляется сравнение дат создания BIOS, записанных в КЧЗП и ПЗУ компьютера. В случае если защищаемая программа будет незаконно скопирована и установлена на компьютер другой серии, то КЧЗП это обнаружит и будут выполнены аварийные действия.

Дата создания BIOS записана в ПЗУ по адресу FFFF:0005 и занимает 8 байтов. Программа, позволяющая ее определять, представлена в примере 1.

Пример 1. Программа, позволяющая получить дату создания BIOS

```
#include <stdio.h>
#include <dos.h>
main()
{
char far *ptr; int i;
ptr=MK_FP(0xFFFF,0x0005);
printf("\nДата создания Вашей BIOS: ");
for (i=0; i<8; printf( "%c", *(ptr++)). i++);
}
```

Версия используемой MS DOS

Так как версия операционной системы MS DOS, с которой работает пользователь на данном компьютере, не меняется на протяжении достаточно длительного времени, то ее также можно использовать в качестве параметра для организации проверки в КЧЗП. Этот контроль желательно проводить в комплексе с другими методами защиты, например, дополнительно к проверке даты создания BIOS, что еще более суживает класс машин, на которых может работать защищаемая программа.

Программа, определяющая версию работающей MS DOS, представлена в примере 2. Она использует прерывание 21h DOS, в котором функция 30h возвращает требуемое значение в регистрах AL (0 для DOS 1.x) и AH.

Пример 2. Программа определения версии MS DOS

```
#include <stdio.h>
#include <dos.h>
main()
{
struct WORDREGS {unsigned int
AX,BX,CX,DX,SI,DI,CFLAG,FLAGS;};
struct BYTREGS {unsigned char
AL,AH,BL,BH,CL,CH,DL,DH;};
union REGS {struct WORDREGS x; struct BYTREGS h;}
in,out;

in.h.AH=0x30;
int86(0x21,&in,&out);
printf("*** Версия используемой MS DOS: %d.%d
***\n",out.h.AL,out.h.AH);
}
```

Тип компьютера

Для обеспечения работы защищаемой программы только на компьютере одного клона надо, чтобы она могла определять его тип. Такая информация содержится в байте, расположенном по адресу FFFF:000E в ROM BIOS со следующей кодировкой (см. табл. 1):

PC - FF; XT - FE,FB; PCjr - FD; AT - FC; PS/2 - FC,FA,F8;
PC-совместимый - F9.

Эта же информация заносится в КЧЗП, и в процессе выполнения программы осуществляется сравнение идентификаторов типа компьютера, записанных в ПЗУ и КЧЗП.

Программа, приведенная в примере 3, позволяет определять тип компьютера, прочитав байт кодировки в ПЗУ.

Пример 3. Программа определения типа компьютера

```
#include <stdio.h>
#include <dos.h>
main()
{
char far *ptr;
ptr=MK_FP(0xF000,0xFFFE);
printf("*** Кодовый байт равен: %x ***\n", *ptr);
}
```

Конфигурация системы и типы составляющих ее устройств

В развитие идеи, изложенной выше, рассмотрим способ, при котором КЧЗП в качестве параметра использует всю конфигурацию системы. Конечно надо учитывать, что этот способ будет давать ошибки и отказывать в выполнении законной копии программы, если какие-либо составляющие вычислительной системы будут изменены (например, отключен принтер), либо наоборот законная и незаконная копии программы будут работать на системах с одинаковой конфигурацией.

Рассмотрим способы определения параметров системы, которые используются затем в КЧЗП.

В IBM XT за связь с периферией отвечает микросхема Intel 8255, которая сообщает об установке переключателей на системной плате, принимает ввод с клавиатуры, управляет рядом периферийных устройств, включая микросхему таймера 8253. Мы не останавливаемся на описании работы с ней (подробности изложены в [1]).

Для IBM AT-параметры системы хранятся в так называемой CMOS памяти, построенной на микросхеме MC146818 фирмы Motorola. (Микросхема 8255 в IBM AT отсутствует). Она имеет 64 однобайтовых регистра, пронумерованных от 00 до 3Fh. Назначения некоторых регистров представлены в табл. 2.

Регистр 10h содержит информацию о НГМД: его биты 7-4 описывают накопитель A, а биты 3-0 - накопитель B. Аналогично сведения о НЖМД помещаются в регистр 12h. Контроль этих байтов может быть осуществлен в КЧЗП.

Еще одной характеристикой, которую можно учитывать в КЧЗП, может явиться обьем установленной оперативной памяти и ее расширение. Такие сведения находятся в регистрах 15h-16h, 17h-18h и 30h-31h.

Информация о периферии хранится в байте 14h CMOS памяти. Для ее получения надо сначала записать номер регистра (в данном случае 14h) в порт с адресом 70h, а затем прочитать содержимое регистра через порт 71h. (Для записи какого-то значения в регистр CMOS памяти надо занести адрес регистра в порт с адресом 70h, а значение - в порт 71h.). Значения битов регистра 14h представлены в табл. 3.

В примере 4 приведена процедура определения наличия математического сопроцессора для компьютера IBM AT по байту 14h CMOS памяти. Работа с остальными регистрами CMOS памяти и их битами аналогична.

Компьютер может быть снабжен различными типами дисководов для гибких дисков и винчестеров. Рассмотрим, с помощью

Пример 4. Фрагмент программы определения наличия математического сопроцессора

```
outportb(0x70,0x14);
if ( inportb(0x71) & 2 — 2 )
printf ("*** Имеется математический сопроцессор ***\n");
```

Таблица 2. Назначение некоторых регистров CMOS памяти

РЕГИСТР	НАЗНАЧЕНИЕ
00h	Секунды
01h	Second Alarm
02h	Минуты
03h	Minute Alarm
04h	Часы
05h	Hour Alarm
06h	День недели
07h	День месяца
08h	Месяц
09h	Год
10h	Типы НГМД A: и B: (0000 - не установлен; 0001 - 360Кбайтов; 0010 - 1.2Мбайта)
12h	Типы НЖМД C: и D: - для AT; D: - для PS/2
14h	Байт установленной периферии
15h-16h	Размер памяти на основной системной плате (0100h - 256Кбайтов; 0200h - 512Кбайтов; 0280h - 512Кбайтов+128Кбайтов на плате расширения)
17h-18h	Размер памяти канала ввода/вывода
30h-31h	Размер дополнительной памяти сверх 1Мбайта

каких средств можно получить информацию об их характеристиках кроме доступа к CMOS памяти.

Прерывание 21h DOS компьютера AT имеет функции 32h и 36h, связанные с определением характеристик установленных накопителей. Функция 36h сообщает текущие сведения о доступном пространстве на диске, номер которого загружается в регистр DL. Она возвращает:

- число секторов на один кластер в регистре AX;
- число незанятых кластеров - в BX;
- число байтов в одном секторе - в CX;
- число кластеров на диске - в DX.

Таблица 3. Назначение битов 14h-го регистра CMOS памяти

БИТ	СОДЕРЖИМОЕ	ЗНАЧЕНИЕ
7-6	00/01	Один/два НГМД
5-4	11/01/10	Видеорежим (монохромный/цветной 40*25/цветной 80*25)
3-2	Не используется	
1	1	Имеется математический сопроцессор
0	0/1	Нет/имеется НГМД

Теперь легко узнать, каков объем диска, номер которого помещался в регистр DL. Для этого достаточно вычислить произведение значений регистров AX,CX,DX.

Функция 32h позволяет получить таблицу с параметрами накопителя, номер которого загружен в регистр DL. Ее адрес будет содержаться в регистрах DX:BX. Подробное описание байтов этой таблицы можно найти в любом справочнике по прерываниям.

Состав установленного оборудования проверяется при загрузке, и результат проверки помещается в регистр статуса. Этот регистр занимает два байта, начиная с адреса 0040:0010, и в табл. 4 представлены значения его битов.

Для доступа к этому регистру кроме прямого обращения по адресу можно воспользоваться прерыванием 11h BIOS, которое возвращает два байта его значений в регистре AX.

В языке Turbo C имеется специальная функция BIOSEQUIP из библиотеки стандартных функций <bios.h>, которая возвращает целое число, описывающее оборудование, входящее в систему. Возвращаемое значение интерпретируется набором битовых полей, как представлено в табл. 4.

Еще одно прерывание BIOS – 15h через функцию C0h позволяет получить адрес в ПЗУ, определяемый регистрами ES:BX, по которому находится табл. 5.

Таблица 4. Регистр состава установленного оборудования

БИТ	СОДЕРЖИМОЕ	ЗНАЧЕНИЕ
0	0/1	Нет/есть НГМД
1	0/1	Нет/есть математический сопроцессор 80x87
2-3	11	Оперативная память 64 Кбайта (в AT не используется и всегда равно 11)
4-5	11/01/10	Начальный видеорежим (монохромный/цветной 40*25/ цветной 80*25)
6-7	00/01/10/11	Число НГМД, если бит 0=1 (соответственно 1,2,3,4)
8	XT/AT не используется (наличие микросхемы DMA)	
9-11	Число адаптеров коммуникации RS232	
12	1	Есть игровой адаптер (в AT не используется)
13	XT/AT не используется	
14-15	Число присоединенных принтеров	

В примере 5 приводится текст программы, позволяющей получить доступ к этой таблице.

В настоящее время существуют четыре основных типа адаперов дисплея, которые могут быть установлены в компьютере: монохромный, цветная графическая карта CGA, усовершенствованный графический адаптер EGA и графическая матрица VGA. Они могут быть установлены отдельно, либо могут присутствовать два адаптера, например монохромный и CGA. Рассмотрим, как определить наличие конкретного типа адаптера.

Активность того или иного адаптера производится установкой определенного значения битов 5 и 4 в ячейке памяти по адресу 0040:0010 в области данных ROM BIOS (см. табл. 4). Их комбинации имеют следующее соответствие:

11 - монохромный адаптер;
10 - графический адаптер в режиме 80*25;
01 - графический адаптер в режиме 40*25;
00 - EGA.

Однако если установлен адаптер EGA, то биты 4;5 могут быть и отличными от 00 (в зависимости от состояния переключателей самого EGA). Поэтому сначала необходимо установить факт

Таблица 5

БАЙТЫ	ЗНАЧЕНИЕ
0-1	Число байтов в таблице
2	Код модели компьютера (см. выше)
3	Различие между AT и XT/286 (подмодель)
4	Номер ревизии BIOS
5	80h - 3-й канал DMA, используется BIOS; 40h - второй контроллер прерываний i8259 установлен; 20h - таймер реального времени установлен; 10h - int15h/AH=4Fh вызывается перед int 9h; 8h - допустимо ожидание внешнего события; 4h - расширение BIOS размещено в 640 Кбайтах; 2h - шиной является Micro Channel вместо шины ISA; 1h - резерв
	Замечание: 1/10/86 XT BIOS возвращает некорректное значение 5-го байта

наличия EGA. Для этого необходимо прочитать байт по адресу 0040:0087, биты которого имеют следующие значения:

0-й - 1=эмуляция курсора возможна;
1-й - 1=EGA присоединен к монохромному дисплею;
2-й - 1=ожидание для перетрассировки;
0=разрешена запись видео RAM в любое время;
3-й - 1=EGA не активна;
5-6-й - суммарная RAM на карте EGA (00=64к; 01=128к; 10=192к; 11=246к);
7-й - старший бит видеорежима (1=экран не очищается).

Если этот байт равен 0, то EGA отсутствует, в противном случае

▼ Пример 5. Программа доступа к таблице 5

```
#include <stdio.h>
#include <dos.h>
main()
{
int i,j;
struct WORDREGS {unsigned int
AX,BX,CX,DX,SI,DI,CFLAG,FLAGS;};
struct BYTREGS {unsigned char
AL,AH,BL,BH,CL,CH,DL,DH;};
struct SREGS {unsigned int  ES,CS,SS,DS;} segr;
union REGS {struct WORDREGS x; struct BYTREGS h;}
in,out;

in.h.AH=0xC0;
int 86x(0x15,&in,&out,&segr);
ptr=MK_FP(segr.ES, OUT.x.BX);
for (i=0,j=0; j<6; printf("*** Байт %1d равен: %x
***\n",j,*(ptr+i)),i++,j++);
}
```

EGA установлен. При отсутствии EGA активность монохромного или цветного адаптера определяется по значению 4,5 битов байта по адресу 0040:0010. Если EGA присутствует, то его активность определяется по значению бита 3-го байта по адресу 0040:0087 (когда он равен 0, то EGA является активным адаптером, а когда он равен 1, то активен второй адаптер). Процедура определения активного видеоадаптера, установленного в компьютере, представлена в примере 6.

▼ Пример 6. Определение активного адаптера

```
#include <stdio.h>
#include <dos.h>
main()
{
char far *ptr;
char y;
ptr=MK_FP(0x0040,0x0087);
if ( *ptr==0 )                    /* если EGA нет */
{ ptr=MK_FP(0x0040,0x0010);              /* читаем
                          байт статуса оборудования */
y= (*ptr) & 48;                   /* выделяем биты 4 и 5 */
if (y==48) printf("Монохромный адаптер"); /*  00110000 */
if (y==32) printf("Цветной адаптер 80*25"); /*  00100000 */
if (y==16) printf("Цветной адаптер 40*25"); /*  00010000 */
}
else                              /* если EGA есть */
if ( *ptr & 8 ==0 ) printf("Активен EGA");
}
```

Если активен адаптер EGA, то проверка наличия второго адаптера (монохромного или цветного) осуществляется следующим образом. Сначала надо указать на регистр адреса курсора микросхемы 6845 путем посылки значения 0Fh в порт 3B4h для монохромной карты или 3D4h для цветной карты. После этого читаем и сохраняем адрес курсора через порт 3B5h (3D5h). Теперь посылаем какое-либо значение в порт 3B5h (3D5h), а затем читаем значение из этого же порта. При наличии монохромной (цветной) карты вернется то же значение, в противном случае - другое. И наконец, восстанавливаем прежнее значение регистра адреса курсора. Процедура, демонстрирующая эти действия, показана в примере 7.

▼ Пример 7. Проверка наличия монохромной карты, когда активен EGA или CGA

```
OUTPORT(0x3B4, 0x0F);     /*адрес регистра курсора */
x= INPORT(0x3B5);       /*чтение и сохранение регистра*/
OUTPORT(0x3B5, 100 );     /*посылаем в регистр*/
                          /*любое значение*/
if ( INPORT(0x3B5)!=100 ) ... /*если карта есть*/
                          /* вернется тоже */
OUTPORT(0x3B5,x);         /*восстановление значения*/
                          /* регистра*/
```

Для определения типа дисплея надо проверить бит номер 1 байта, находящегося по адресу 0040:0087. Когда этот бит равен 1 - подсоединяется монохромный дисплей, а когда он равен 0 - цветной. Если установлен цветной монитор, то можно также определить, присоединен ли дисплей IRGB или R'G'B'RGB (улучшенный цветной дисплей IBM). Это определяется установкой четырех переключателей на карте EGA. Установка этих переключателей возвращается в регистре CL при обращении к функции 12h прерывания 10h. Цепочка четырех младших битов должна быть 0110 для улучшенного цветного дисплея.

Та же самая функция сообщает и наличие памяти на карте EGA. Она возвращает кодовое значение в регистре BL (0 - для 64 Кбайтов, 1 - для 128 Кбайтов, 2 - для 192 Кбайтов и 3 - для полных 256 Кбайтов памяти дисплея).

Теперь рассмотрим вопрос, связанный с введением ограничений либо на время, в течение которого может использоваться программа, либо на количество ее запусков.

Ограничение срока использования программ

Иногда изготовители программного обеспечения накладывают ограничения на срок его работы у определенного пользователя, то есть как бы выдают временную лицензию, по истечении срока которой происходит либо самоуничтожение программы, либо она перестает работать.

Идея, лежащая в основе этого метода, достаточно прозрачна: нужно только получить текущую дату из таймера, а затем, сравнив с "лицензионным" сроком использования, либо разрешить работу программы, либо запретить. Однако и недостаток данного метода защиты ясен: можно просто изменить дату, установленную в таймере, и тогда программа будет выполняться незаконно.

Пример простого модуля, проверяющего дату, приведен в листинге 1.

Использование счетчика числа запусков программы

Теперь рассмотрим, как можно организовать счетчик, который бы следил за количеством запусков защищаемой программы. Довольно часто такой счетчик устанавливают в:

- CMOS памяти (для IBM AT);
- специальных числовых файлах;
- boot секторе дискеты.

Давайте рассмотрим эти возможности. (Заметим, что CMOS память и boot сектор используются и для хранения метки для КЧЗП).

Счетчик в CMOS памяти IBM AT

В отличие от ОЗУ CMOS память питается от автономного источника (батарейки), и поэтому ее содержимое не уничтожается после отключения компьютера. Как было отмечено выше, CMOS RAM состоит из 64 однобайтовых регистров, пронумерованных от 00h до 3Fh, из которых 1-й, 3-й и 5-й байты можно задействовать для нашей задачи. В системе эти байты предусмотрены для использования прерываниями, для которых в них заносится лимит времени, в течение которого они должны быть осуществлены. Однако реально такая необходимость возникает крайне редко.

Первый регистр будем использовать в качестве индикатора того, что в CMOS память занесен счетчик, а третий регистр - для хранения значения счетчика запусков программы. Максимальная величина такого счетчика может быть не более 255-60=195 (максимальное целое число, помещающееся в регистре, минус число различных значений временного интервала).

Программа, приведенная в листинге 2, инициирует счетчик числа запусков защищаемой программы, т.е. она заносит в первый регистр CMOS памяти метку (0x60), а в третий - непосредственно значение счетчика. Записываемые в эти регистры значения заведомо не могут находиться в этих регистрах и поэтому не могут интерпретироваться как системные. Метка необходима для предотвращения использования значения, которое может изначально находиться в третьем регистре, без инсталляции счетчика, а также как параметр КЗЧП для предотвращения незаконного копирования.

После того как программа инициализации отработала, может запускаться сама защищаемая программа, модуль КЧЗП которой приведен на листинге 3. КЧЗП производит проверку инициализации счетчика по наличию метки и, если она установлена и лимит числа запусков еще не исчерпан, разрешает выполнение программы, а затем модифицирует значение счетчика.

Счетчик в BOOT секторе

Загрузочный сектор дискеты (винчестера) занимает 1-й сектор на 0-й дорожке 0-й стороны. Его байты с 4 по 11 содержат информацию, которую записывает программа форматирования (например такую, как "MS DOS 3.3"). Она не несет какой-либо системной нагрузки, и поэтому вполне может быть изменена.

Программа инсталляции, представленная на листинге 4, выполнает инициализацию счетчика в boot секторе. Она записывает в 4-й байт признак того, что будет использоваться счетчик (символ "Y"), а в 5-й - значение счетчика.

Теперь может запускаться защищаемая программа, КЧЗП которой приведена на листинге 5. Она проверяет метку в четвертом байте и, если это "Y", выполняется собственно программа, а затем уменьшается счетчик. Когда счетчик обнуляется, в четвертый байт заносится метка "N" и при повторных попытках запуска защищаемой программы КЧЗП не допускает ее выполнения.

Счетчик в специальном файле

Описанные выше два способа организации счетчика в CMOS памяти и boot секторе теряют смысл, если необходимо организовать работу нескольких программ, причем каждую со своим контролем числа произведенных запусков. Поэтому рассмотрим метод,в котором при инсталляции защищаемой программы на диске происходит создание специального числового файла, в который и заносится счетчик числа запусков. Такой файл организуется с префиксом имени защищаемой программы и расширением "g", а его атрибуты имеют значение "файл только для чтения" и "скрытый". Программа, реализующая эти процедуры, представлена на листинге 6. После трансляции ее вызов должен производиться с полным названием защищаемой программы в качестве параметра.

КЧЗП представлена на листинге 7. Сначала она проверяет наличие файла-счетчика. В случае его отсутствия программа прерывается, в противном случае - контролируется само значение счетчика и, если оно не нулевое, происходит выполнение собственно защищаемой программы и декрементирование счетчика. ■

ЛИТЕРАТУРА

1. **Р. Джордейн. Справочник программиста ПК типа IBM PC XT/AT. - М: Финансы и статистика, 1991.**

Журнал "Монитор" приглашает к сотрудничеству:

- Дилеров по распространению журнала
- Специалистов по анализу состояния компьютерного рынка
- Авторов оригинальных статей по рограммированию и вычислительной технике

Тел. (095)238-99-33
Факс (095)238-32-78
Адрес для писем: 117419, Москва, а/я 765.

Анонс

"Монитор" – версия 8.92

В. Водолазкий. Коммерческие алгоритмы шифрования: основные алгоритмы и их реализация. Часть 2.

Описание системы распределения ключей Дифи-Хелмана и системы с открытым ключом RSA.

Л. Ливеровский. Модельный файл Clarion

Использование модельных файлов Clarion для расширения возможностей Дизайнера.

М. Иринчук. Проблема памяти и адаптер VGA

Использование видеопамяти для хранения данных.

Д.Правиков.Замедление работы программ

Рассказывается о методах искуственного замедления скорости выполнения программ.

Д. Солдатенков. Имеющий уши да услышит

Звуковые эффекты в программе на Turbo Pascal.

С. Трепыхалин. Несколько полезных макросов для Turbo Pascal 6.0

Применение компилятора макрокоманд для облегчения ввода текста программ.

А. Ермолов. Что такое IPX и как его использовать в своих целях. Часть 2.

Рассказывается о создании резидентных программ, поддерживающих сетевые протоколы обмена данными.

А.Кельманзон.Особенности C++ в компиляторах Borland

В статье рассказывается об отличии Borland C++ 2.0 и 3.0 от классического C++, описанного в книге Б.Страуструпа.

А. Сидоров. Оптимизация в Zortech C++

Сравнительный анализ качества оптимизации наиболее распространенных компиляторов с языка С.

Д. Рогаткин, А. Федоров. Программирование в среде Windows 3.1

Первая статья из серии, посвященной программированию в среде Windows 3.1.

Листинг 1. Программа проверки предельной даты использования программы

```
#include <stdio.h>
#include <dos.h>
main()
{
struct DATE {int da_year; char da_day; char da_mon;} today;
int day = 31,
month = 12,
year = 1992;

getdate(&today);
printf("*** Сегодня %2d.%2d.%4d ***\n",
today.da_day,today.da_mon,today.da_year);
printf("*** Программу разрешено использовать до %2d.%2d.%4d
***\n",
day,month,year);
if ( compare(day,month,year,today) = = 1 )

/*-----------------------------------*/
/*                                   */
/*   ТЕЛО ЗАЩИЩАЕМОЙ ПРОГРАММЫ      */
/*                                   */
/*-----------------------------------*/
printf ("*** Программа выполнена ***\n");
else printf("*** Время использования программы истекло ***\n");
}

compare(day1,mon1,yea1,today)
int day1,mon1,yea1;
struct DATE {int da_year; char da_day; char da_mon;} today;
{
int day2,mon2,yea2;
day2 = today.da_day;
mon2 = today.da_mon;
yea2 = today.da_year;
if ( yea2 > yea1 ) return(0);
if ( yea2 < yea1 ) return(1);
if ( mon2 > mon1 ) return(0);
if ( mon2 < mon1 ) return(1);
return(day2 < = day1);
}
```

Листинг 2. Программа инсталляции счетчика в CMOS память

```
#include <stdio.h>
#include <dos.h>
main()
{
char count;
printf("*** Введите значение счетчика:"); scanf("%d",&count);
CMOS_SET(1,0x60);
CMOS_SET(3,60+count);
printf("*** CMOS инсталлирована ***\n");
}

/* функция записи значения cmos в регистр adress CMOS памяти */
CMOS_SET(adress,cmos)
char adress, cmos;
{
outportb( 0x70, adress );
outportb( 0x71, cmos );
}
```

Листинг 3. Программа работы со счетчиком, находящимся в CMOS памяти

```
#include <stdio.h>
#include <dos.h>
main()
{
int ind;
if (CMOS_GET(1) = = 0x60)
{
ind = CHANGE_COUNT();
if (ind = = 0) printf("*** Лимит запусков программы исчерпан ***\n");
else /*-----------------------------------*/
/*                                   */
/*   ТЕЛО ЗАЩИЩАЕМОЙ ПРОГРАММЫ      */
/*                                   */
/*-----------------------------------*/
printf ("*** Программа окончена ***\n");
}
}

/* функция изменения значения счетчика */
CHANGE_COUNT()
{
char count;
count = CMOS_GET(3)-60;
if ( count = = 0 ) return(0);
count - = 1;
CMOS_SET(3,60+count);
if (count ! = 1 )
printf ("*** Осталось %d запусков***\n",count);
else printf ("*** Внимание, остался один запуск! ***\n\007");
return(1);
}

/* функция получения значения из регистра adress CMOS памяти */
CMOS_GET(adress)
char adress;
{
char cmos;
outportb( 0x70, adress );
cmos = inportb(0x71);
return(cmos);
}
```

Листинг 4. Программа инсталляции счетчика в boot сектор

```
#include <stdio.h>
#include <dos.h>
main()
{
char count;
char sector[512];

printf("*** Введите значение счетчика: ***\n"); scanf("%d",&count);
READ_SECTOR(sector);
sector[3] = 'Y';
sector[4] = count;
WRITE_SECTOR(sector);
printf("*** Счетчик инициализирован ***\n");
}

/* Процедура чтения сектора */
READ_SECTOR(sector)
char sector[512];
{
struct WORDREGS {unsigned int AX,BX,CX,DX,SI,DI,CFLAG,FLAGS;};
struct BYTREGS  {unsigned char AL,AH,BL,BH,CL,CH,DL,DH;};
struct SREGS    {unsigned int ES,CS,SS,DS;} segr;
union REGS {struct WORDREGS x; struct BYTREGS h;} in,out;

in.h.AH = 0x02;                 /* Номер функции чтения     */
in.h.AL = 1;                    /* Число секторов для чтения */
in.h.CH = 0;                    /* Номер дорожки */
in.h.CL = 1;                    /* Номер сектора */
in.x.DX = 0;                    /* 0-я сторона и драйвер A: */
segr.ES = FP_SEG(sector);
in.x.BX = FP_OFF(sector);
int86x(0x13,&in,&out,&segr); /* Чтение сектора       */
}

/* Процедура записи сектора */
WRITE_SECTOR(sector)
char sector[512];
{
struct WORDREGS {unsigned int AX,BX,CX,DX,SI,DI,CFLAG,FLAGS;};
struct BYTREGS {unsigned char AL,AH,BL,BH,CL,CH,DL,DH;};
struct SREGS {unsigned int ES,CS,SS,DS;} segr;
```

```
union REGS {struct WORDREGS x; struct BYTREGS h;} in,out;

in.h.AH = 0x03;                 /* Номер функции записи     */
in.h.AL = 1;                    /* Число секторов для записи */
in.h.CH = 0;                    /* Номер дорожки */
in.h.CL = 1;                    /* Номер сектора */
in.x.DX = 0;                    /* 0-я сторона и драйвер A:  */
segr.ES = FP_SEG(sector);
in.x.BX = FP_OFF(sector);
int86x(0x13,&in,&out,&segr); /* Запись сектора */
}
```

Листинг 5. Программа проверки счетчика в boot секторе

```
#include <stdio.h>
#include <dos.h>
main()
{
char count;
char sector[512];

READ_SECTOR(sector);
if ( sector[3] ! = 'Y' )
{ printf("*** Ваша программа не выполнялась ***\n");
exit(255); }
sector[4] - = 1;
count = sector[4];
if (count ! = 1 )
printf ("*** Осталось %d запусков программы ***\n",count);
else { sector[3] = 'N';
printf ("*** Внимание, остался один запуск! ***\n\007"); }
WRITE_SECTOR(sector);

/*-------------------------------------*/
/*                                     */
/*   ТЕЛО ЗАЩИЩАЕМОЙ ПРОГРАММЫ        */
/*                                     */
/*-------------------------------------*/
printf ("*** Программа окончена ***\n");
}
```

Листинг 6. Программа инсталляции счетчика в специальный файл

```
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
char *p,*name;
int ls,i,count;
FILE *f;

p = argv[1];                    /* Получаем длину  */
while (*p) p + +;               /* имени защищаемой */
ls = p-argv[1];                 /* программы       */

name = malloc(ls);              /* Получаем имя защищаемой */
i = 0; while ( *(name + i) = *(argv[1] + i) ) i + +;
                                /* программы в name      */

name[ls-3] = 'g';               /* Присваиваем числовому  */
name[ls-2] = '\0';              /* файлу расширение .g    */

f = fopen(name,"w");
printf("*** Введите значение счетчика: ***"); scanf("%d",&count);
fprintf(f,"%d",count);
fclose(f);
HIDDEN_READONLY(1,name);

printf("*** Счетчик инсталлирован в файл %s ***\n",name);
}

/* Процедура установки атрибутов файла */
HIDDEN_READONLY(sw,path)
int sw;
char path;
{
struct WORDREGS {unsigned int  AX,BX,CX,DX,SI,DI,CFLAG,FLAGS;};
struct BYTREGS  {unsigned char AL,AH,BL,BH,CL,CH,DL,DH;};
struct SREGS    {unsigned int  ES,CS,SS,DS;} segr;
union REGS {struct WORDREGS x; struct BYTREGS h;} in,out;

in.h.AH = 0x43;                 /* выбираем функцию 43h      */
in.h.AL = 0;                    /* Читаем байт атрибутов файла */
in.x.DX = FP_OFF(path);         /* DS:DX указывает на файл    */
segr.DS = FP_SEG(path);
int86x(0x21,&in,&out,&segr);    /* Получаем в CX атрибуты файла */
if (sw = = 1)
out.h.CL = out.h.CL ! 3;        /* Устанавливаем бит 0 и 1    */
else out.h.CL = out.h.CL & 0;   /* Сбрасываем все биты        */
in.h.AH = 0x43;                 /* Выбираем функции 43h       */
in.h.AL = 1;                    /* Устанавливаем байт атрибутов */
in.h.CL = out.h.CL;
int86(0x21,&in,&out,&segr);
```

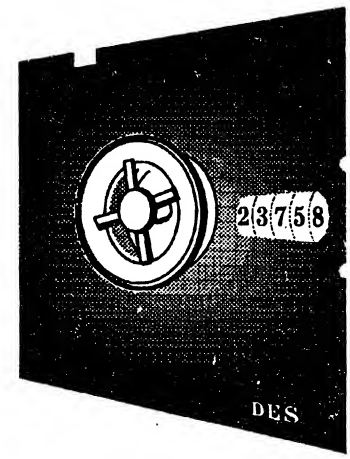
Листинг 7. Программа работы со счетчиком в специальном файле

```
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
char *p,*name;
int ls,i,count;
FILE *f;

p = argv[0];                    /* Получаем длину */
while (*p) p + +;               /* имени защищаемой */
ls = p-argv[0];                 /* программы */

name = malloc(ls);              /* Получаем имя защищаемой */
i = 0; while ( *(name + i) = *(argv[0] + i) ) i + +;
                                /* программы в name */

name[ls-3] = 'g';               /* Присваиваем числовому */
name[ls-2] = '\0';              /* файлу расширение .g */

f = fopen(name,"r");
if ( f = = NULL )
{ printf("*** Ваша программа не выполнялась ***\n");
exit(255); }
HIDDEN_READONLY(0,name);
CHANGE_COUNT(f,name);
HIDDEN_READONLY(1,name);
/*-------------------------------------*/
/*                                     */
/*   ТЕЛО ЗАЩИЩАЕМОЙ ПРОГРАММЫ        */
/*                                     */
/*-------------------------------------*/
printf ("*** Программа окончена ***\n");
}

/* Процедура изменения значения счетчика */
CHANGE_COUNT(f,path)
char path;
FILE *f;
{
int count;

fscanf (f,"%d",&count);
fclose(f);
if ( count = = 0 )
{ printf ("*** Дальнейшее использование программы невозможно
***\n");
exit(255);}
count - = 1;
f = fopen(name,"w");
fprintf (f,"%d",count);
fclose(f);
printf ("*** Осталось %d запусков ***\n",count);
}
```

Коммерческие системы шифрования: основные алгоритмы и их реализация

Часть 1

В. Водолазкий



Какую цель преследовали вы, читатель, купив этот номер журнала? По всей видимости, скажете вы, найти в нем какие-нибудь материалы, которые можно было бы использовать в работе, а может быть даже подзаработать. И вдруг начинает разыгрываться воображение: увидит ваш продукт фирма Microsoft или, на худой конец, Borland International и обалдеет! И получите вы гору "капусты", и купите себе IBM PS/2 mod.80-111, и будете целыми днями играть в Space Quest IV или в Strip Poker.

Вы совершенно правы! Вы купили именно тот журнал, который вам необходим. Мы открываем новую рубрику "Как это делается". В ней мы будем рассказывать о тех проблемах, которые пользуются спросом на нашем формирующемся рынке, и по возможности описывать алгоритмы для их реализации. Начиная с этого номера, мы опубликуем серию статей, посвященных реализации программного обеспечения для закрытия коммерческой информации. Мы будем не просто приводить программы, а детально опишем наиболее мощные на сегодняшний день алгоритмы шифрования данных, используемые в коммерческих системах : DES, RSA, система распределения ключей Диффи-Хелмана, ElGamal и др.

Стандарт шифрования данных Data Encryption Standard

В 1977 году Национальное бюро Стандартов США (NBS) опубликовало стандарт шифрования данных Data Encryption Standard (DES), предназначенный для использования в государственных и правительственных учреждениях США для защиты от несанкционированного доступа важной, но несекретной информации. Алгоритм, положенный в основу стандарта, распространялся достаточно быстро, и уже в 1980 году был одобрен ANSI. С этого момента DES превращается в стандарт не только по названию (Data Encryption Standard), но и фактически. Появляются программное обеспечение и специализированные микро-ЭВМ, предназначенные для шифрования/дешифрования информации в сетях передачи данных и на магнитных носителях. К настоящему времени DES является наиболее распространенным алгоритмом, используемым в системах защиты коммерческой информации. Более того реализация алгоритма DES в таких системах является просто признаком хорошего тона! За примерами далеко ходить не надо. Программа DISKREET из пакета Norton Utilities, предназначенная для создания зашифрованных разделов на диске, использует именно алгоритм DES. "Собственный алгоритм шифрования" отличается от DES только числом итераций при шифровании. Почему же DES добился такой популярности?

Основные достоинства алгоритма DES:
– используется только один ключ длиной 56 битов;
– зашифровав сообщение с помощью одного пакета, для расшифровки вы можете использовать любой другой;
– относительная простота алгоритма обеспечивает высокую скорость обработки информации;
– достаточно высокая стойкость алгоритма.

К сожалению, на русском языке алгоритм DES до сих пор не описывался. Автору, по крайней мере, удалось найти только одну статью в журнале "В мире науки" (он же Scientific American), из которой удалось понять только то, что DES имеет очень много общего с рецептом приготовления ирландского рагу [1].

Поэтому автор считает, что необходимо заполнить данный пробел и дать возможность программистам (в том числе и авторам вирусов) попробовать себя в реализации по-настоящему серьезных продуктов. Дерзайте, Кулибины и Павличуки!

Итак, впервые на русском языке - полное описание американского стандарта шифрования коммерческой информации DES.

DES осуществляет шифрование 64-битовых блоков данных с помощью 56-битового ключа. Дешифрование в DES является операцией обратной шифрованию и выполняется путем повторения операций шифрования в обратной последовательности (несмотря на кажущуюся очевидность, так делается далеко не всегда. Позже мы рассмотрим шифры, в которых шифрование и дешифрование осуществляются по разным алгоритмам).

Процесс шифрования заключается в начальной перестановке битов 64-битового блока, шестнадцати циклах шифрования и, наконец обратной перестановки битов (рис.1).

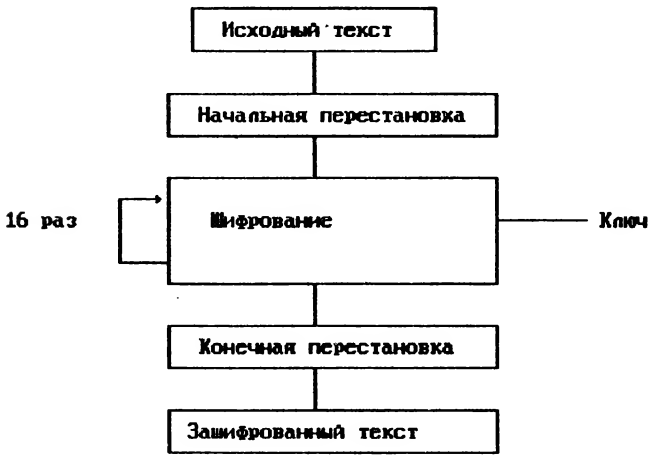


Рис.1. Общая схема процесса шифрования в алгоритме DES

Необходимо сразу же отметить, что ВСЕ таблицы, приведенные в данной статье, являются СТАНДАРТНЫМИ, а следовательно должны включаться в вашу реализацию алгоритма в неизменном виде. Все перестановки и коды в таблицах подобраны разработчиками таким образом, чтобы максимально затруднить процесс расшифровки путем подбора ключа. Кроме того, стандарт есть стандарт. Не уподобляйтесь МЕНАТЕПу, запустившему в мутные воды нашего рынка модем Lexand-2400, якобы "адаптированный к отечественным линиям связи". ("Адаптация" этого изделия состоит в том, что, "вооруженный" совковой идеологией и этикой, модем при работе умудряется пролезть в соседние каналы и обеспечить себе высокую "пробивную" способность" за счет других абонентов сети, т.е. расталкивая всех остальных пользователей локтями. При этом, хотя ваш канал наполовину занят Lexand, платить будете все равно вы) [2].

Но вернемся к нашему алгоритму.

Пусть из файла считан очередной 8-байтовый блок T, который преобразуется с помощью матрицы IP (рис.2) начальной перестановки, что даст в результате:

T0 = IP(T).

58	50	42	34	26	18	10	02
60	52	44	36	28	20	12	04
62	54	46	38	30	22	14	06
64	56	48	40	32	24	16	08
57	49	41	33	25	17	09	01
59	51	43	35	27	19	11	03
61	53	45	37	29	21	13	05
63	55	47	39	31	23	15	07

Рис.2. Матрица начальной перестановки IP

Затем шестнадцать раз повторяется процедура шифрования этого блока с помощью функции f. Пусть T_i обозначает результат i-й итерации. Тогда предположим, что

$L_i = t_1 ... t_{32}$, (первые 32 бита) $R_i = t_{33} ... t_{64}$ (последние 32 бита),

т.е.

$T_i = L_i R_i$.

Тогда результатом i-й итерации будет:

$L_i = R_{i-1}$, $R_i = L_{i-1} + f(R_{i-1}, K_i)$,

где + - операция ИСКЛЮЧАЮЩЕЕ ИЛИ;

K_i - i-е преобразование ключа шифрования.

Функция f выполняет операции над значением R_{i-1} (результатом прошлой операции и текущим значением 48-битового ключа K_i (с отсечением лишних битов). Кстати, после 16-й итерации левая и правая половины блока местами не меняются (рис. 3).

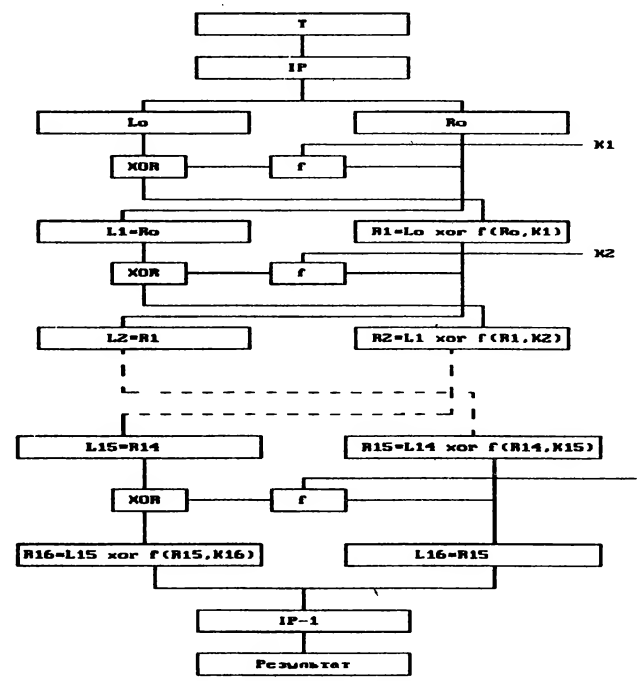


Рис.3. Общая блок-схема шифрования DES

По окончании шифрования осуществляется восстановление позиций битов с помощью матрицы перестановок IP^{-1} (рис.4).

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Рис. 4. Матрица обратной перестановки IP^{-1}

Теперь рассмотрим, что же скрывается под преобразованием, скрытым буквой f.

Функция f и S-матрицы

На каждой итерации массив R_{i-1} с помощью таблицы распределения E (рис.5) расширяется до 48 битов.

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Рис. 5. Таблица распределения битов E

Полученный результат (обозначим его $E(R_{i-1})$) складывается по модулю 2 (операция XOR) с текущим значением ключа K_i и затем разбивается на восемь 6-битовых блоков $B_1...B_8$.

		Номер столбца																
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Номера строк	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S1
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S2
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S3
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	S4
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	S5
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	S6
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S7
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S8
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

Рис.6. Матрицы преобразования 6*4 S1..S8

То есть:

$E(R_{i-1}) + K_i = B_1B_2...B_8$.

Далее каждый из этих блоков используется как номер элемента в матрицах S1...S8, содержащих четырехбитовые значения (рис. 6).

В результате, применив операцию выбора к каждому из блоков, мы получим

$S1(B_1)S2(B_2)S3(B_3)...S8(B_8)$

Этот 32-битовый блок (матрицы S_j - содержат 4-битовые значения) преобразуется с помощью матрицы перестановки P (рис. 7):

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Рис. 7. Матрица перестановки битов P

Таким образом,

$f(R_{i-1}, K_i) = P(S1(B_1),...S8(B_8))$

Общая схема данного участка алгоритма представлена на рис. 8.

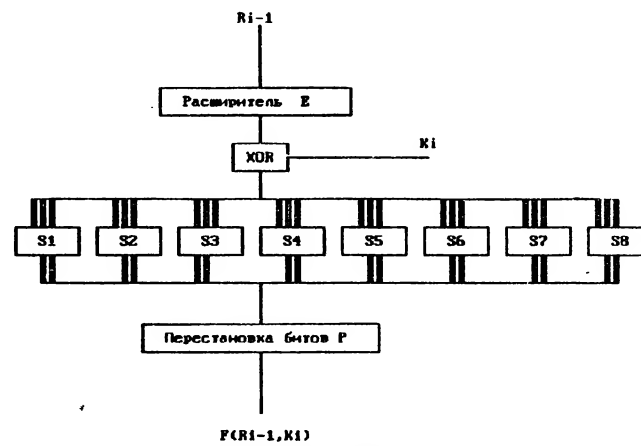


Рис.8. Вычисление функции F(Ri, Ki)

Следует отметить, что выбор элемента в матрице S_j осуществляется достаточно оригинальным образом. Пусть на вход поступает 6-битовый блок B_j = b1b2b3b4b5b6, тогда двухбитовое число b1b6 выбирает строку матрицы, а b2b3b4b5 - номер столбца. Результатом $S_j(B_j)$ будет 4-битовое число, расположенное по указанному адресу.

Например, если B_3=100110, тогда необходимо извлечь значение, размещенное в 3-м столбце второй строки, что составит 9 или 1001.

Как вы уже заметили, на каждой итерации используется новое значение ключа - K_i. Теперь самое время поговорить о том,

Как вычисляется значение ключа

На каждой итерации используется новое значение ключа K_i, которое вычисляется из начального ключа K. K представляет собой 64-битовый блок с восемью битами контроля по четности, расположенными в позициях 8,16,24,32,40,48,56,64.

Для удаления контрольных битов и подготовки ключа к работе используется матрица PC^{-1} (рис.9).

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Рис. 9. Матрица первоначальной подготовки ключа

Результат преобразования $PC^{-1}(K)$ разбивается на две половины C_0 и D_0 по 28 битов каждая. После этого блоки C_0 и D_0 на каждой итерации последовательно сдвигаются влево. Пусть C_i и D_i обозначают значения, полученные на i-й операции:

$C_i = LS_i(C_{i-1})$ $D = LS_i(D_{i-1})$,

где LS_i представляет собой i-й элемент матрицы сдвига LS (табл.1).

Табл. 1. Таблица сдвигов для вычисления ключа

Номер итерации	Сдвиг (бит)
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Полученное значение вновь "перемешивается" в соответствии с матрицей PC-2 (рис. 10).

Таким образом:

$K_i = PC\text{-}2(C_iD_i)$

Блок-схема алгоритма вычисления i-й итерации ключа приведена на рис.11.

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Рис. 10. Матрица завершающей обработки ключа

Два слова о дешифровании

Восстановление исходного текста осуществляется по этому алгоритму, но (!) вначале вы используете ключ K15, затем - K14 и так далее. Теперь вам должно быть понятно, почему автор настойчиво рекомендует использовать приведенные матрицы. Если вы начнете самовольничать, вы, должно быть, получите очень секретный шифр, но вы сами не сможете его потом раскрыть!

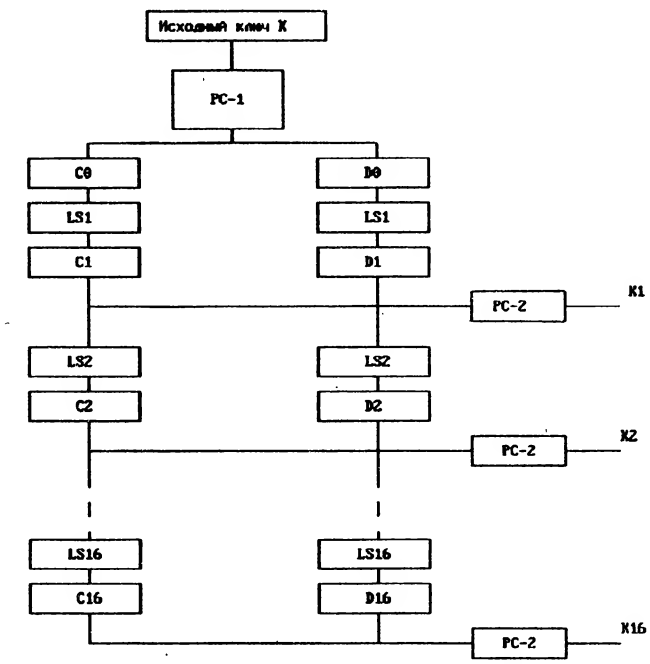


Рис.11. Блок-схема алгоритма вычисления ключа

Режимы работы алгоритма DES

Для наиболее полного удовлетворения всем требованиям, предъявляемым к коммерческим системам шифрования, реализованы несколько режимов работы алгоритма DES. Наиболее широкое распространение получили режимы:

- электронный шифрблокнот (Electronic Codebook) - ECB;
- цепочка цифровых блоков (Cipher Block Chaining) - CBC;

– цифровая обратная связь (Cipher Feedback) - CFB;
– внешняя обратная связь (Output Feedback) - OFB.

Кстати, если вы написали программу защиты данных и хотите дать полноценную рекламу, то автор рекомендует прямо указывать, какие из режимов поддерживает ваше детище. Это, вообще говоря, признак хорошего тона на рынке программного обеспечения средств защиты. Нет смысла раскрывать весь алгоритм, вы просто указываете : DES-CBC или DES-CFB и, как говорится, "умный догадается, а дураку и не надо".

Давайте рассмотрим перечисленные выше режимы

DES-ECB

Длинный файл разбивается на 64-битовые отрезки (говоря, по-русски, по 8 байтов) (рис.12). Каждый из этих блоков кодируется независимо с использованием одного и того же ключа шифрования. Основное достоинство этого алгоритма - простота реализации. Недостаток - относительно слабая устойчивость против квалифицированных криптоаналитиков.

В частности, не рекомендуется использовать данный режим работы для шифрования EXE файлов, потому что первый же блок - заголовок файла, является вполне удачным началом для взлома всего шифра.

В то же время следует признать, что этот режим в силу своей простой реализации наиболее популярен среди любительских разработок.

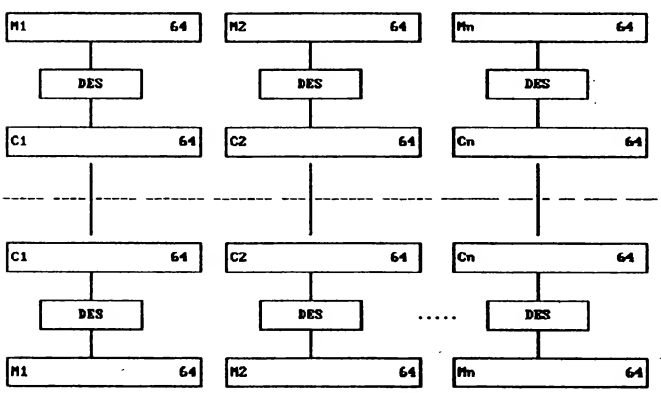


Рис.12. Работа алгоритма DES в режиме ECB

DES-CBC

В этом режиме (рис.13) исходный файл M разбивается на 64-битовые блоки : $M = M_1M_2...M_n$. Тогда для всех i = 1,N (N - число блоков) результат шифрования Ci будет определяться следующим образом:

C_0 = начальное значение

$C_i = DES(M_i \text{ XOR } C_{i-1})$

Дешифрование выполняется в обратном порядке.

C_0 = начальное значение

$M_i = C_{i-1} \text{ XOR } DES(C_i)$

Прелесть данного режима состоит в том, что он не позволяет накапливаться ошибкам при передаче. Блок M_i является функцией только C_{i-1} и C_i. Поэтому ошибка при передаче приведет к потере только двух блоков исходного текста.

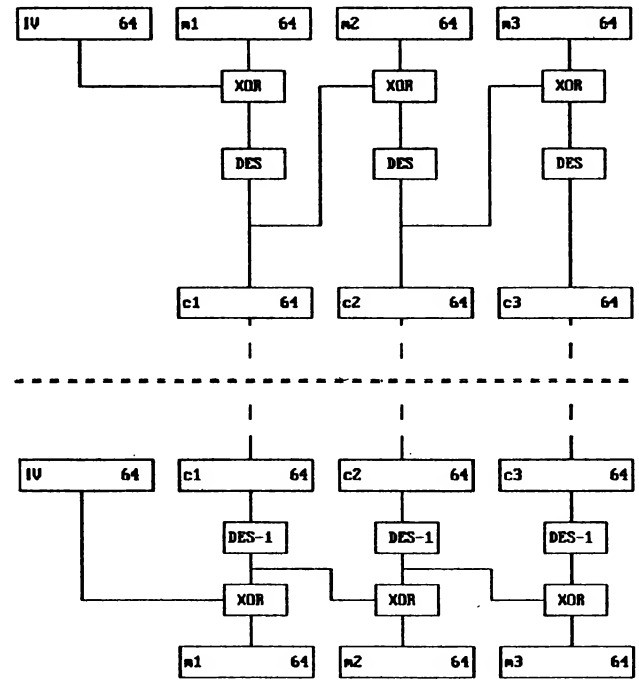


Рис.13. Работа алгоритма в режиме CBC

DES-CFB

В этом режиме размер блока отличается от 64 Файл, подлежащий шифрованию (дешифрованию), считывается последовательными блоками длиной t битов (t = 1,64). 64-битовый регистр сдвига инициализируется строкой S0, которая может быть либо заполнена пробелами, либо в нее записывается исходное значение ключа K. Предположим, что в результате разбиения на блоки, мы получили N блоков, длиной по t битов каждый (остаток, как вы догадываетесь, дописывается нулями или пробелами). Тогда для любого i = 1,N :

$C_i = M_i \text{ XOR } P_i$,

где P_i обозначает t старших битов предыдущего зашифрованного блока. Обновление сдвигового регистра осуществляется путем удаления его старших N битов и дописывания C_i справа.

Восстановление зашифрованных данных также не представляет труда: P_i и C_i вычисляются аналогичным образом и

$M_i = C_i \text{ XOR } P_i$.

Блок-схема режима CFB приведена на рис.14.

DES-OFB

Режим OFB также использует переменный размер блока и сдвиговый регистр, инициализируемый аналогично режиму CFB. Но в этот раз для каждого сеанса шифрования данных необходимо использовать новое начальное состояние регистра S0, которое должно пересылаться по каналу открытым текстом.

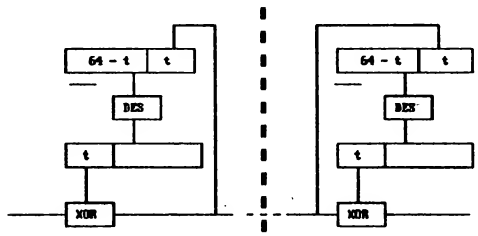


Рис.14. Работа алгоритма DES в режиме CFB

Положим $M = M_1M_2...M_n$. Для всех $i = 1,N$,
$C_i = M_i$ хог P_i,
где P_i - старшие t битов операции $DES(S_{i-1})$. Отличие от режима CFB состоит в методе обновления сдвигового регистра. В данном случае это осуществляется путем отбрасывания старших t битов и дописывания справа P_i (рис. 15).

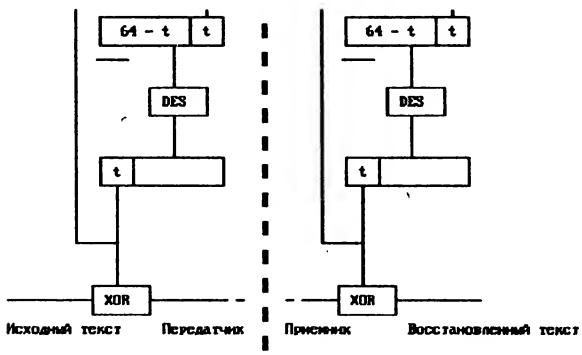


Рис.15. Блок-схема алгоритма DES в режиме OFB

Итак, мы рассмотрели алгоритм шифрования коммерческой информации DES. Автор надеется, что материал данной статьи поможет читателям (хотя бы некоторым) в преодолении финансовой ямы и явится одним из стимулов к появлению на нашем рынке полноценных систем защиты информации.

Но исчерпывает ли DES все аспекты защиты данных? Пожалуй нет, ведь по-прежнему остается одно весьма уязвимое место: ключ. Тот, кто владеет ключом от шифра, без труда может расшифровать сообщение. Казалось бы, это принципиальное ограничение систем шифрования. Но дотошные математики разработали алгоритмы шифрования с разными ключами для шифрования и дешифрования, которые положили основу широко распространенным сегодня системам с открытым ключом.

Но о них речь пойдет в следующем номере. ■

Литература

1. Джером К. Джером. Трое в одной лодке, не считая собаки. - М.: Учпедгиз, 1958.
2. КомпьютерПресс, 1991, N 8.

Скрытая мощь старенького EPSON

Желание написать эту программу возникло у меня в тот момент, когда я скручивал в рулон выползшую из принтера распечатку программы на ассемблере строк примерно в 800. Главное неудобство при работе на любом из языков программирования, по моему убеждению, состоит в том, что, начиная с некоторого уровня сложности, любая программа становится необозримой. Один из аспектов этой необозримости состоит в чисто физическом неудобстве обращения с длинными листингами программ. Это неудобство наиболее выраженно проявляется при попытке разобраться с программой, пока едешь с работы или на работу в нашем переполненном транспорте. Да и просто жалко переводить бумагу на отладочную распечатку.

Но, к счастью, принтеры EPSON далеко ушли от своего предка - телетайпа и позволяют программировать основные режимы их функционирования. Настройка принтера осуществляется путем пересылки в него специальных управляющих последовательностей.

Что такое управляющая последовательность? Как правило, принтер осуществляет печать символов, передаваемых ему из ЭВМ в коде ASCII. Однако, как вы, вероятно, знаете, коды ASCII с 0 по 32 относятся к так называемым управляющим кодам и не выводятся на печать. Вот они-то и используются для настройки режимов работы принтера.Всего принтеры фирмы EPSON содержат более 70 команд [1], которые могут быть разделены на следующие группы:

А) Команды управления принтером:
- инициализация,
- программная активация, деактивизация,
- настройка скорости печати,
- настройка на используемый тип бумаги,
- включение звукового сигнала принтера,
- программирование межстрочного интервала.

Б) Команды управления данными:
- возврат каретки,
- перевод строки,
- забой и др.

В) Форматирование страницы:
- вертикальная и горизонтальная табуляция,
- вертикальные и горизонтальные границы,
- режим выравнивания текстовых строк (заслуживает отдельного рассмотрения).

Г) Выбор стиля печати:
- режимы NLQ и DRAFT,
- ширина символов,
- печать со смещением,
- подчеркивание,
- двойной удар,
- режимы верхнего и нижнего индексов,
- переключение кодовых таблиц.

Д) Вывод на печать графических изображений.

При этом наиболее часто используется последовательность кодов, начинающихся с символа <Esc> (код ASCII-27), за которым следуют другие символы, определяющие код операции.

Так, для включения верхнего индекса, например, показателя степени, используется управляющая последовательность : <Esc>+'S'+Chr(0).

Это формальное описание управляющей последовательности легче всего пояснить программой на Паскале, приведенной в примере 1.

Пример 1. Настройка режима функционирования принтера из программы на TurboPascal

```
Program SetSuperscript;
CONST ESCAPE = Chr (27);
uses Printer;
BEGIN
Write (Lst, Escape, 'S', Chr(0));
END.
```

Для включения нижнего индекса, например, для отображения основания логарифма, используется последовательность: Esc+'S'+Chr(1), а для возврата к нормальному режиму печати: <Esc>+'T'.

Примеры:

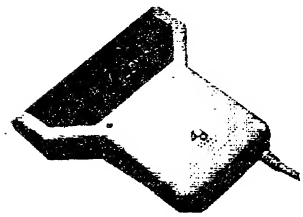
2^{24}, $\log_{10}(a^2+b^2)$.

стр.24 ▶

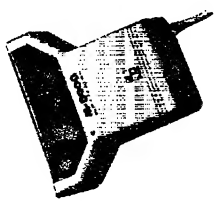
Системы и комплексы

Фирма "Системы и комплексы", официальный дистрибьютер фирмы Mustek Optic GmbH, представляет:

Mustek™



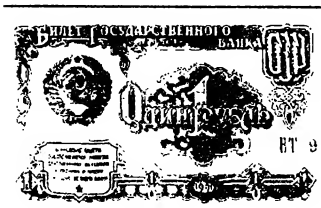
Полутоновый ручной сканер **Matador 105** (Mustek Optic GmbH) прост, надежен и удобен в обращении. **Matador 105** может быстро сканировать графические изображения, рисунки, диаграммы, чертежи, иллюстрации или тексты. С его помощью легко оформить брошюры, рекламные листки, отчеты, приглашения или письма. Характеристики сканера **Matador 105**: ширина сканирования - 105мм; разрешающая способность - 400, 300, 200, 100 точек на дюйм; количество уровней серого тона - 64; количество шкал полутонов - 3; контраст - подстраиваемый; размеры - 138x136x35 мм; вес - 300 гр.



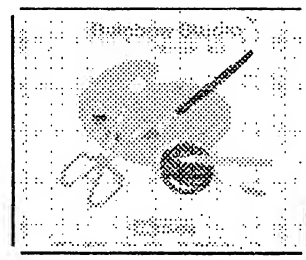
Серый ручной сканер **Gray Artist** (Mustek Optic GmbH) отличается от полутонового **Matador**'а более высоким качеством сканируемого изображения (подобно тому, как черно-белый телевизор отличается от монохромного монитора). **Gray Artist** может сканировать в чисто серой шкале (256 цветов) при разрешении до 400 точек на дюйм. Кроме того, **Gray Artist** позволяет вести сканирование и в полутоновом режиме, где разрешение достигает 800 точек на дюйм. **Gray Artist** обладает переключателями, изменяющими размер полутоновой ячейки. Этот сканер значительно мощнее хорошо зарекомендовавшего себя на нашем рынке сканера MARS 800.

Цветной ручной сканер **Color Artist** (Mustek Optic GmbH) подходит как для любителей, так и для профессионалов. В то время, как цветные ручные сканеры других фирм-изготовителей различают лишь 4,096 цветов, **Color Artist** оперирует с 262,144 цветами. **Color Artist** использует для подсветки высокочастотную флюоресцирующую лампу, что позволяет даже при сканировании в монохромном режиме получать изображение по качеству близкое к получаемому с планшетных сканеров. **Color Artist** дает возможность получать не только цветное изображение, но и изображения в чисто серой шкале (256 чисто серых цветов), в полутоновом и штриховом режимах. Таким образом, при увеличившемся качестве оцифровки изображения сканер **Color Artist** объединяет в себе возможности сканеров **Gray Artist** и **Matador 105**. В цветных режимах ведется сканирование с разрешениями от 50 до 200 точек на дюйм, в чисто сером, полутоновых и штриховом режимах - от 100 до 400 точек на дюйм.

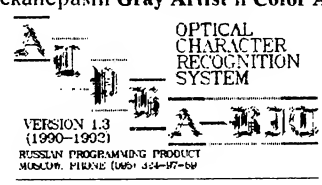
На ВСЕ сканеры мы даем гарантию!!!



Пакеты **Scankit Gray** и **Scankit Color** (Mustek Optic GmbH), поставляемые вместе со сканерами **Matador 105, Gray Artist, Color Artist**, дают возможность сканировать, вырезать, печатать и сохранять в файле графические изображения в форматах наиболее популярных графических пакетов: (.TIF), (.CUT). (.PCX), (.IMG), (.MSP), (.TGA). Одно из достоинств **Scankit** - возможность склеивать две полосы изображения на визуальном уровне. В версии **Scankit Gray,** поставляемой вместе со сканером **Gray Artist**, имеется возможность регулировать яркость и контрастность серого изображения. **Scankit Color** также позволяет регулировать насыщенность цветного изображения и манипулировать яркостью и контрастностью отдельно по каждой составляющей RGB-сигнала; кроме того, можно использовать специальные алгоритмы редукции цветного изображения для повышения качества вывода на цветной принтер.



Мощный многооконный (до 10 окон) пакет полной обработки изображений **Rainbow Studio** (VI&C Technology Inc.) позволяет сканировать, считывать, записывать, редактировать и выводить на печать (в том числе и с цветоделением) черно-белые, серые и цветные изображения в форматах TIFF и PCX. Пакет предоставляет набор средств для работы с прямоугольными, овальными и произвольными элементами изображений, обладает возможностями графического редактора, осуществляет поворот и масштабирование изображений. Помимо этого пакет **Rainbow Studio** позволяет манипулировать яркостью, контрастностью, насыщенностью, цветовым решением изображения, при этом уровень контрастности может быть нелинейным. **Rainbow Studio** предоставляет возможность конвертирования серых изображений в цветные, полутоновые, штриховые, и наоборот. Если Вы приобрели **Rainbow Studio** вместе с **Gray Artist**, то серое изображение Вы можете "раскрасить" и отретушировать, выбрав при этом необходимую Вам цветовую палитру. Пакет **Rainbow Studio** поставляется вместе со сканерами **Gray Artist** и **Color Artist**.



ALPHA-BIT (КНПП "БИТ") Обучаемый пакет оптического распознавания текста, который уже обучен разнообразным шрифтам и устойчиво работает как с ручными, так и с планшетными сканерами. Процесс обучения происходит очень быстро. **ALPHA-BIT** позволяет импортировать графические файлы в форматах TIFF и PCX, распознает как латиницу, так и кириллицу. Предусмотрены три режима обучения, автоматическое разбиение страницы на зоны с последующей автокомпоновкой. Пакет обладает современным интерфейсом, который полностью поддерживает манипулятор "мышь". **ALPHA-BIT** практически по всем параметрам (включая скорость и точность распознавания) превосходит пакет Mars Pro Reader, которым ранее комплектовались сканеры фирмы **Mustek GmbH**. Пакет поставляется как отдельно, так и вместе со сканерами *(со значительной скидкой)*.

ORFO ("Информатик") - резидентная программа контроля правописания русского текста. Обладая словарем в 200 000 основ, **ORFO** "узнает" до 3 млн. словоформ. Она также позволяет проверять согласование слов в предложении, выявляя ошибки в роде, числе, падеже и т.д. Если **ORFO** не знает какого-либо слова, Вы имеете возможность добавить его в словарь. При этом **ORFO** будет распознавать все формы нового слова. Вы можете работать с **ORFO** в любом текстовом редакторе, не выходя из него!

Кроме **ORFO** Вы можете приобрести спеллер **ПРОПИСЬ** (НПК "АГАМА"), эквивалентный по возможностям, но работающий в среде Microsoft Windows.

CopyPCX (Концерн "Новинтех") - резидентная программа, предназначенная для копирования графического изображения с экрана ПК (режимы EGA и VGA) в дисковый файл формата PCX графического редактора **PC PaintBrush**. **CopyPCX** позволяет выводить изображение, записанное в файле в формате PCX на экран. **CopyPCX** практически независима от применяемых пользователем графических редакторов!

Для Вас:

ВЫСОКОКАЧЕСТВЕННЫЕ СКАНЕРЫ
ФИРМЕННЫЕ ПРОГРАММЫ
ПРЕДПРОДАЖНЫЕ КОНСУЛЬТАЦИИ
РАЗУМНЫЕ ЦЕНЫ
ГАРАНТИЯ и СОПРОВОЖДЕНИЕ

Наш адрес: РФ, 119517, Москва, ул.Нежинская 13, Фирма "Системы и комплексы" Тел: **(095) 441-1100** Факс: **(095) 442-5792**

Мирное применение вирусной технологии

О. Сурин

По мере дальнейшего развития программных и компьютерных технологий все острее ставится проблема компьютерного пиратства. Необузданность отечественного рынка лишний раз подтверждает уязвимость программных продуктов перед лицом нелегальных копировщиков. Создание средств, обеспечивающих защиту от несанкционированного использования по существу превратилось в отдельную отрасль индустрии программ. Естественно ожидать, что не существует какой-либо универсальной эффективной защиты от пиратства, так как массовое распространение такой защиты резко увеличивает для нее шанс подвергнуться взлому. Отсюда понятно желание программистов иметь под рукой несложное перенастраиваемое средство, которое легко можно изготовить в "домашних условиях".

В приводимой статье автор предлагает оригинальный метод для защиты файлов типа .EXE от нелегального запуска. Идея этого метода навеяна желанием имплантировать в исполняемый файл некий "доброкачественный" вирус, который сможет отслеживать количество запусков программы и в случае превышения этого количества некоторого заданного значения приводить к разрушению файла. При использовании такой защиты нет необходимости привязываться к каким-либо аппаратным особенностям PC, а сама процедура состоит лишь в запуске приводимой ниже программы. Возможность защиты распространяется на EXE файлы, работающие в операционной системе DOS (версия 2.0 и выше).

Для реализации такого механизма предлагается поместить в защищаемый файл заранее подготовленный блок, полученный ассемблированием файла **defender.asm**. В момент запуска защищенного файла управление будет передаваться на установленный блок и после выполнения всех нужных действий обратно в начало программы. Для первоначальной передачи управления можно изменить несколько слов в заголовке EXE файла, которые DOS использует для установки регистров CS и IP. Чтобы проиллюстрировать сказанное, рассмотрим заголовок EXE файла более подробно. Эта структура (табл.1) предваряет все стандартные файлы типа .EXE и содержит в себе следующие элементы (см. заголовок EXE файла):

Таблица1. Структура заголовка EXE файла в MS-DOS

Смещение	Содержание	Описание
0h	4Dh,5Ah	Идентификатор EXE файла ('MZ')
2h	PartPage	Длина неполной последней страницы
4h	PageCnt	Длина модуля в 512-байтовых страницах (включая заголовок)
6h	ReloCnt	Число элементов в таблице перемещений
8h	HdrSize	Размер заголовка в 16-байтовых параграфах
0Ah	MinMem	Минимальное количество памяти, требуемое за концом программы в параграфах
0Ch	MaxMem	Максимальное количество памяти, требуемое за концом программы в параграфах
0Eh	ReloSS	Смещение сегмента стека (для установки SS)
10h	ExeSP	Значение регистра SP при запуске
12h	ChkSum	Контрольная сумма слов в файле
14h	ExeIP	Значение регистра IP при запуске
16h	ReloCS	Смещение кодового сегмента (для установки CS)
18h	TableOff	Смещение первого элемента в таблице перемещений
1Ah	Overlay	Номер загружаемого оверлея (0 для основного модуля)
1Ch		Начиная с этого адреса располагается таблица перемещаемых элементов

Когда мы набираем в командной строке имя исполняемого EXE файла и нажимаем клавишу 'ENTER', DOS производит следующие действия.

1. Создает префикс сегмента программы (PSP - Program Segment Prefix). PSP представляет собой 100h-байтовую область памяти, вслед за которой за-

гружается программа. В эту область заносятся системные данные, которые бывает полезно считывать ; в частности по адресу 80h в PSP заносится командная строка, сопутствующая программе (табл.2).

Таблица 2. Структура PSP

Смещение	Размер	Описание
0	2	Команда завершения INT 20h
2	2	Номер последнего доступного блока памяти (номер параграфа)
4	1	Зарезервировано
5	5	Команда вызова диспетчера функций DOS
0a	4	Адрес завершения программы
0e	4	Адрес обработчика критических ошибок
16	16h	Зарезервировано для использования DOS
2c	2	Адрес сегмента окружения DOS
2e	2eh	Зарезервировано для использования DOS
5c	10h	Используется для обработки первого параметра командной строки с помощью FCB (*)
6c	14h	Используется для обработки второго параметра командной строки с помощью FCB
80h	1	Длина строки параметров (**)
81h	7fh	Содержимое командной строки
100h		Начинается размещение программы
(*)	FCB (file control block) - блок управления файлом	
(**)	Используется также как смещение начала области обмена данных с диском (DTA - disk transfer area) по умолчанию	

2. Считывает первые 1ch байтов заголовка в локальную область памяти и на основе этой информации определяет размер загрузочного модуля как (PageCnt - 1) * 512 + PartPage - HdrSize * 16.

3. Вычисляет смещение этого модуля в исходном файле = HdrSize * 16.

4. Считывает загрузочный модуль в память, начиная с адреса
START_SEG : 0000, где START_SEG = PSP +10h

5. Для каждого элемента таблицы перемещений пересчитывает ссылки таким образом, чтобы обеспечить работу программы в любом месте памяти.

6. Выделяет для программы память согласно MaxMem и MinMem.

7. Присваивает начальные значения регистрам :

```
ES = DS = PSP,
SS = START_SEG + ReloSS,
SP = ExeSP,
CS = START_SEG + ReloCS,
IP = ExeIP.
```

8. Передает управление по адресу CS:IP.

Таким образом адрес, с которого начинается выполнение программы, однозначно определяется комбинацией регистров CS:IP. Для нашей цели необходимо изменить эти значения в соответствующих полях заголовка, чтобы они указывали на конец файла, где размещается **defender** блок [1]. Новое значение CS можно посчитать как количество целых 16-байтовых параграфов загружаемого модуля, а новое значение IP как 16 минус дополнение до полной длины модуля. Однако прямолинейное решение задачи оказывается неудачным по следующей причине.

Для доступа к ячейкам памяти процессор выбирает значение сегмента данных из регистра DS. Как видно из листинга 1, содержимое DS приравнивается регистру CS с помощью команд

```
MOV AX,CS
MOV DS,AX
```

т.е. командный сегмент одновременно является и сегментом данных. Следовательно корректное обращение к переменным внутри блока будет осуществлять-

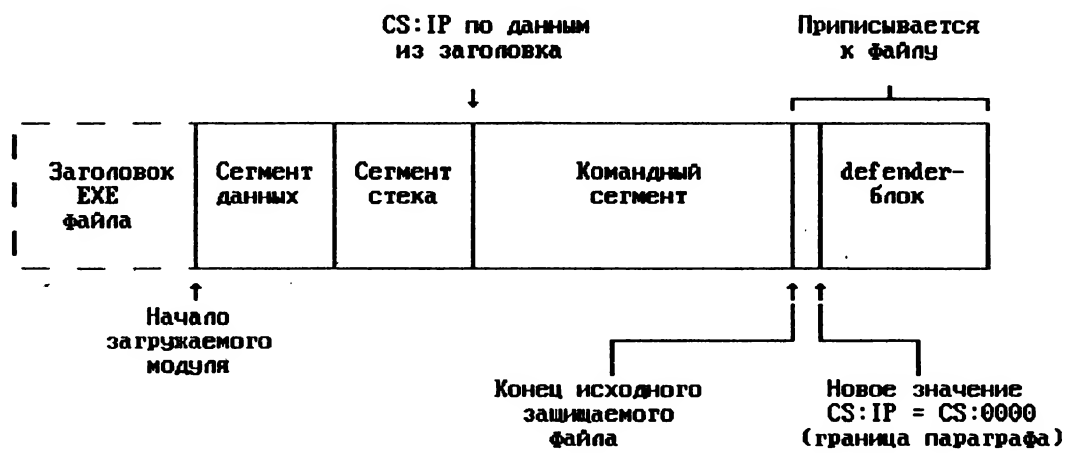


Рис. 1. Защищаемый файл и установка защитного блока

ся только в том случае, если начало сегмента данных будет совпадать с началом блока. Когда мы компилируем defender.asm, то ассемблер "подразумевает" это автоматически. Однако в случае записи защитного блока в конец исходного файла сегмент данных, совпадающий с командным сегментом, будет начинаться с адреса CS:0000, а сам блок с адреса CS:IP. Таким образом все обращения к переменным внутри defender блока будут ошибочны на величину смещения, равную IP.

К счастью, существует простой и изящный способ преодоления этой трудности. Обратимся к рис. 1.

Вместо того чтобы стараться учесть добавку смещения, мы просто положим ее равной нулю, а defender блок разместим с границы очередного параграфа, следующего после конца защищаемого файла. Теперь когда сегменты команд, данных и сам блок начинаются с одной точки, проблем с адресацией больше не возникнет. Отметим, что расположение сегментов данных и стека на рис.1 является совершенно произвольным. Сегменты могут располагаться в других частях исполняемого модуля или могут отсутствовать вообще.

Легко видеть, что между границей модуля и приписанным блоком всегда существует некоторое "межсегментное пространство", если только длина модуля не кратна шестнадцати. Так как мы не имеем возможности записывать данные за пределы файла, то для установки defender блока нам необходимо сначала заполнить это пространство чем-нибудь. Программно это заполнение реализуется путем записи перед блоком последовательности символов 90h, что преследует чисто отладческие цели. В машинном представлении символу 90h соответствует команда холостого хода (NOP). Поэтому если в результате какого-либо ошибочного перехода мы попали в межсегментный заполнитель, сбоя системы не произойдет.

Programmer

"Доброкачественные" вирусы можно использовать для защиты файлов типа .EXE от нелегального запуска

Перейдем к процессу записи defender блока. Исходным материалом является файл defender.exe, полученный компиляцией и компоновкой defender.asm[2]. Для передачи необходимых параметров в defender.exe служит процедура WRITE_TO_BLOCK (листинг 2). Так как первые 512 байтов (смещение относительно начала файла 0..511) занимают EXE заголовок, то с помощью оператора

```
SEEK ( defender,512 )
```

указатель положения файла устанавливается на первый символ исполняемого кода (конечно же, пропускать заголовок или устанавливать указатель на первый символ необязательно).

После считывания кода в символьный массив chblock в последние 8 байтов вписываются количество запусков до саморазрушения, значения CS и IP, с которых должна выполняться исходная защищаемая программа, а также длина блока. После этого кодовая часть файла defender.exe перезаписывается. Напомним, что данные в двухбайтовых словах, с которыми оперирует IBM PC, располагаются "вверх ногами" или "задом наперед", т.е. действует принцип: младший байт - по младшему адресу, старший байт - по старшему адресу. Для приведения слов к такому машинному представлению употребляются функции TURBO PASCAL hi и lo, возвращающие соответственно старший и младший байты аргумента.

В чем же проявляются защитные функции блока defender? Очевидно, в контроле числа запусков защищаемой программы. При этом необходимо считать текущее значение числа запусков, проанализировать его и записать новое значение в защищаемый EXE файл. К сожалению, мы не можем настроить защиту на какое-то определенное имя файла, так как любое его переименование сведет на нет все наши усилия. Для однозначной идентификации запущенной программы обратимся к так называемому окружению DOS (DOS environment), которое еще иногда называется средой DOS.

Мы будем использовать информацию, хранящуюся в окружении DOS, как это изложено в [1], что позволит нам при активизации блока защиты автоматически определять имя запущенной программы и полный путь исполняемого файла.

После локализации пути загрузки соответствующий файл открывается и в него (в последние два байта) записывается новое, уменьшенное на единицу, число возможных санкционированных запусков. Если это количество равно нулю, принимаются меры для приведения файла в негодность.

Итог наших ухищрений с окружением : defender блок всегда найдет программный файл на диске, где бы он ни располагался и как бы ни назывался.

После того как defender блок заканчивает свои действия, управление переходит в основной модуль. Для вычисления абсолютного сегмента адреса возврата к содержимому csret прибавляются значение текущего сегмента PSP и его длина в параграфах. Это объясняется тем, что значение csret, скопированное из соответствующего поля заголовка, представляет собой лишь файловое смещение сегмента команд, выраженное в параграфах. Для учета реального местоположения в памяти к этому смещению необходимо добавить адрес "начала программы", каковым и является адрес (PSP + 10h). Содержимое ipret, соответствующее начальному значению IP, является относительным счетчиком команд внутри сегмента и поэтому пересчету не подлежит.

Структура defender блока довольно проста и читатель несомненно в ней разберется самостоятельно.

Отметим еще только, что переход управления из блока обеспечивает оператор

```
JMP DWORD PTR CS:[SI],
```

где [si] - указывает на смещение и сегмент адреса возврата.

Подчеркнем, что именно сложность и запутанность defender блока определяют способность защиты противостоять нелегальному вскрытию. Применение таких средств, как дополнительная кодировка информации, изменение прерывания пошагового исполнения, проверка правильности записи в файл и т.д. существенно усложняет работу взломщиков. Впрочем на этом поприще читателю предоставляется возможность проявить свою фантазию и творческие способности в полной мере.

Внедрение защитного блока в файл осуществляет процедура WRITE_TO_FILE - основная часть рассматриваемой программы (листинг 2). В массив слов заголовка chint заносятся изменяемые параметры и соответствующие символы записываются в файл.

Одним из преимуществ (на взгляд автора предлагаемого метода) является возможность быстрой и простой отмены защиты с ранее защищенного файла. Эту функцию выполняет процедура ESC_DEFEND. Из информационной части defender блока считываются координаты адреса начала выполнения и переписываются в заголовок. Помимо этого заново вычисляются значения PageCnt и PartPage, а сам defender блок отсекается с помощью оператора truncate.

Вниманию читателей предлагается также процедура VERIFY, в функции которой входит определение состояния защиты файла. Критерий срабатывания процедуры - наличие или отсутствие специальной метки, расположенной по смещению 14 относительно конца файла. В случае если защита установлена, выводится количество оставшихся возможных запусков.

В заключение еще раз отметим, что предлагаемый метод представляет собой защиту от запуска, но не от копирования. Таким образом, даже если нелегально скопированная программа получит распространение, механизм саморазрушения исключает возможность ее использования. Та легкость, с которой метод позволяет устанавливать и отменять защиту, возможно послужит стимулом для его дальнейшего развития. ■

[1] defender (англ.) - защитник.
[2] Автором использовался Turbo Assembler 1.0 (Borland, 1988).

Литература.

1 В.Водолазкий. Индикатор заражения программы компьютерным вирусом. Монитор, 1992, N 3, стр. 52-53.

▸ *стр. 19*

Кроме того, вы можете сжать индексы, чтобы распечатка стала больше похожа на традиционную запись:

$\log_{10}(a_i^2 + b_j^2)$

Для сжатия необходимо переслать в принтер код $0F, а для восстановления нормального размера - код $12.

Теперь у нас появляется естественный вопрос: что произойдет, если мы просто переключимся в режим верхнего или нижнего (кому как больше нравится) индекса и будем печатать в этом режиме. Как ни странно, но число строк, умещающихся на одном листе, при этом останется прежним.

Все строки в данном абзаце распечатаны в режиме вывода "нижний индекс". Как вы можете убедиться, увеличился зазор между строками (вернее говоря, он остался постоянным, уменьшился размер самой строки).

Что же делать? Выход есть, и очень простой. Необходимо соответствующим образом изменить межстрочный интервал. Принтеры EPSON умеют делать и это! В ходе экспериментов я установил, что оптимальное значение межстрочного интервала для печати шрифтом "индекс" составляет 6/72 дюйма (нет, я не апологет экзотических систем счисления. Дело в том, что все настройки размеров в принтерах буржуазного производства осуществляются именно в дюймах). Для того чтобы настроить соответствующий интервал, вам необходимо передать в принтер управляющий код:

Esc+'A'+Chr(6)

Для использования этих возможностей (за исключением программирования межстрочного интервала), я ввел ряд команд для форматтера текстов редактора Multi-Edit (табл. 1). В результате появилась возможность просто отказаться от неудобного и громоздкого редактора ChiWriter, единственное достоинство которого - возможность работы с математическими формулами. Но если вы не "ругаетесь трехэтажным интегралом", то вполне можно обойтись и обычным редактором.

Таблица 1. Управляющие коды для редактора Multi-Edit V4.0.

Команда форматтера	Описание	Управляющие коды
SL	Включить режим нижнего индекса	127S11
SU	Включить режим верхнего индекса	127S10
SN	Отключить режим индекса	127T
CD	Включить режим сжатия на 60%	115
ND	Выключить режим CD	118
BS	Назад на одну позицию (забой)	18

Команды, приведенные в таблице, очевидны, за исключением, пожалуй, BS. Ее назначение состоит в том, чтобы облегчить пользователю отображение переменных, имеющих одновременно верхние и нижние индексы.

log.cd.sl10.nd.sn(a.cd.sli.bs.su2.sn.nd + b.sl.cdj.bs.su2.nd.sn)

Рис.1. Пример написания формулы $\log_{10}(a_i^2+b_j^2)$ с помощью команд таблицы 1.

На рис. 1 приведен пример использования команд из табл. 1 для написания формулы $\log_{10}(a_i^2+b_j^2)$. Возможно, запись выглядит слишком громоздко, но в Multi-Edit имеется, по меньшей мере, три способа упрощения ввода:

- клавиатурные макросы,
- макрокоманды Multi-Edit,
- пользовательские коды форматтера.

Перечисленные команды предназначены для переключения различных режимов при печати одного файла (например, данной статьи). Для перевода принтера в постоянный режим "экономичной" печати вы можете воспользоваться программой EKONOM.ASM, приведенной в листинге 1. Программа осуществляет аппаратный сброс принтера с помощью вектора прерывания 17h [2], а затем передачу управляющих последовательностей:

- включить режим сжатия,
- включить режим нижнего индекса,
- настроить межстрочный интервал.

стр. 64 ▸

Листинг 1

```
cseg        segment
        assume cs:cseg,ds:cseg,ss:cseg

        mov ah,62h                      ; функция получения адреса PSP
        int  21h                        ; сегмент PSP возвращается в BX
        mov ds,bx
        mov ds,word ptr ds:002ch        ; загрузили в DS сегментный адрес
                                        ; окружения
        mov ax,ds
        mov es,ax               ; загрузили сегмент сканируемой строки
        mov di,0                ; загрузили смещение сканируемой строки
        cld                     ; направление обработки строк - вперед
        mov al,0                ; загрузить код пробела
        mov cx,1000             ; установили счетчик байтов
        repne scasb             ; сканируем до первого пробела
        cmp byte ptr [di],0     ; сравниваем следующий символ
        jne  r                  ; если он нулевой, то строка найдена
        add  di,3               ; пропустить счетчик дополнительных строк

        mov ah,3dh              ; функция DOS: открыть файл
        mov dx,di               ; DS:DX указывает на строку с именем файла
        mov al,2                ; режим чтения и записи
        int  21h                ; описатель файла возвращается в AX
        mov bx,ax               ; сохранили его в BX

        mov ax,cs
        mov ds,ax

        cmp count_exe,0         ; если счетчик запусков равен нулю
        jbe  break              ; ... то запуск несанкционированный
        dec  count_exe

        mov ah,42h ; функция DOS: установить указатель положения файла
        mov al,2        ; ... на конец файла
        mov cx,0
        mov dx,0
        int  21h                ; позиция указателя возвращается в DX:AX
        sub  ax,2               ; вычитаем из AX два байта
        sbb  dx,0               ; ... с учетом возможного заема

        mov cx,dx
        mov dx,ax
        mov ah,42h
        mov al,0                ; переместили указатель на
        int  21h                ; последние два байта

        mov ah,40h              ; функция записи в файл
        mov dx,offset count_exe         ; DS:DX - адрес count_exe
        mov cx,2                ; количество записываемых байтов
        int  21h

        mov ah,3eh              ; функция DOS: закрыть файл
        int  21h

        mov ah,62h              ; получить адрес PSP
        int  21h                ; PSP = = > BX
        mov ax,bx
        add  bx,10h             ; добавить длину PSP
        add  csret,bx           ; вычислить сегмент возврата
        mov si,offset ipret     ; подготовиться к переходу
        mov ds,ax               ; установить начальные значения DS
        mov es,ax               ; и ES
        jmp  dword ptr cs:[si]  ; длинный переход в основную программу
                                ; обработка несанкционированного запуска :
break:
        mov ah,42h              ; установить указатель положения файла
        mov al,0                ; на начало файла
        mov cx,0
        mov dx,0
        int  21h

        mov ah,40h              ; затереть первые 2000 байт
        mov dx,0
        mov cx,2000
        int  21h

metka       db 'DEFENDER'       ; идентификатор блока
blocksize dw 0000h              ; размер блока
ipret dw 0000h                  ; смещение
csret dw 0000h                  ; и сегмент адреса возврата
count_exe dw 0000h              ; счетчик запусков

cseg ends
        end
```

Листинг 2

```
program EXE_DEFENDER;

var  f,block : file of char;
     flag : byte;
     ch90h : char;
     chblock : array [0..1000] of char;
     chint : array [0..$1d] of word;
     hdrsize,exeip,relocs,count_exe : word;
     blocksize : integer;

procedure MESSAGE; begin   writeln;
     writeln('Enter work mode:');
     writeln;
     writeln(' 1 - set defending');
     writeln(' 2 - upset defending');
     writeln(' 3 - verify file for protection');
     writeln;
     readln(flag);
end;

procedure IO_VERIFY;
begin
     if ioresult < > 0 then
     begin
          writeln ('INPUT/OUTPUT ERROR !');
          halt;
     end;
end;

procedure OPEN_FILE;
     var
          filename : string;

begin
          case flag of
     1:   begin
               writeln ('Enter start count:');
               readln(count_exe);
               writeln ('Enter filename to set protection:');
          end;

     2:   begin
               writeln ('Enter filename to UPset protection:');
          end;

     3:   begin
               writeln ('Enter filename to verify file',
                        ' for protection:');
          end;

     else begin
               writeln ('ERROR!');
               halt;
          end; end; { of case }

     {$I-}
     readln (filename);
     assign (f,filename);
     reset (f);
     {$I+}

     io_verify;
end;
```

```
procedure READ_HEADER;

{ Процедура считывает первые $1c символов в байты массива chint и
вычисляет требуемые параметры }

var
     i : integer;
     ch : char;

begin
     for i := 0 to $1c do
          begin
               read (f,ch);
               chint[i] := ord(ch);
          end;

     hdrsize := chint[9] * 256 + chint[8];
     exeip := chint[$15] * 256 + chint[$14];
     relocs := chint[$17] * 256 + chint[$16];
end;

procedure VERIFY;

{ Процедура проверяет наличие идентификатора defender блока в ана-
лизируемом файле }

var
     mstr : string;
     i,count : integer;
     ch,ch1 : char;

begin
     seek (f,filesize(f) - 16);
     mstr := '';
     for i := 1 to 8 do
          begin
               read (f,ch);
               mstr := mstr + ch;
          end;
     if mstr < > 'DEFENDER' then
          begin
               writeln('This file without defending.');
               halt;
          end
     else writeln('This file protected.');

{ Если метка присутствует, читаем последние два байта для нахожде-
ния count_exe : }

     seek(f,filesize(f)-2);
     read (f,ch);
     read (f,ch1);
     count := ord(ch1) * 256 + ord(ch);
     write('Count of unregister starts = ');
     writeln(count);
end;



procedure WRITE_TO_BLOCK ( count_exe : word; spacecnt : integer );

{ Служит для передачи параметров в defender.exe }

var
     i : integer;
     blocksize0 : integer;

begin
     assign (block,'defender.exe');
     reset (block);
     seek (block,512);

{ Длина исполняемой части defender.exe : }
     blocksize0 := filesize(block)-512;
     blocksize := blocksize0 + spacecnt;

     for i := 1 to blocksize0 do
          read (block,chblock[i]);

{ Записываем данные в последние 8 байтов: }
     chblock [blocksize0] := chr(hi(count_exe));
     chblock [blocksize0-1] := chr(lo(count_exe));
     chblock [blocksize0-2] := chr(hi(relocs));
     chblock [blocksize0-3] := chr(lo(relocs));
     chblock [blocksize0-4] := chr(hi(exeip));
     chblock [blocksize0-5] := chr(lo(exeip));
     chblock [blocksize0-6] := chr(hi(blocksize));
     chblock [blocksize0-7] := chr(lo(blocksize));

{ Перезаписываем defender.exe : }
     seek (block,512);
     for i := 1 to blocksize0 do
          write (block,chblock[i]);
end;



procedure ESC_DEFEND;

{ Процедура обеспечивает отмену состояния защиты }

var
     chout : array [0..$1c] of char;
     iploch,iphich,csloch,cshich,ch,
     blocksizelo,blocksizehi : char;
     oldpagecnt,oldpartpage : word;
     i,blocks : integer;

begin
     seek (f,0);
     for i := 0 to $1c do read(f,chout[i]);
     seek (f,filesize(f) - 8);

{ Читаем и вычисляем длину defender блока : }
     read (f,blocksizelo);
     read (f,blocksizehi);
     blocks := ord(blocksizehi) * 256 + ord(blocksizelo);

{ Считываем IP и CS : }
     read (f,iploch);
     read (f,iphich);
     read (f,csloch);
     read (f,cshich);

{ Значения partpage и pagecnt без defender блока : }
     oldpagecnt := (filesize(f) - blocks) div 512 + 1;
     oldpartpage := (filesize(f) - blocks) mod 512;

     chout [$15] := iphich;
     chout [$14] := iploch;
     chout [$17] := cshich;
     chout [$16] := csloch;
     chout [$5] := chr(hi(oldpagecnt));
     chout [$4] := chr(lo(oldpagecnt));
     chout [$3] := chr(hi(oldpartpage));
     chout [$2] := chr(lo(oldpartpage));

{ Переписываем заголовок : }
     seek(f,0);
     for i := 0 to $1c do write(f,chout[i]);

{ Отсекаем defender блок : }
     seek(f,filesize(f)-blocks);
     truncate(f);

     close(f);
end;



procedure PROCESS;

{ Установка defender блока }

var
     i : integer;
     ch : char;
```

```
        size_exe : longint;
        spacecnt,ost : integer;
        partpage,pagecnt,paragrafs_module,
        newcs,newip,pageblock,partblock,
        newpagecnt,newpartpage : word;

begin

{ Определение сегмента расположения defender блока : }

        size_exe := filesize(f);
        paragrafs_module := size_exe div 16;
        newcs := paragrafs_module + 1 - hdrsize;
        newip := 0;

{ Число символов заполнителя : }
        ost := size_exe mod 16;
        spacecnt := 16 - ost;

{ Запись параметров в defender блок : }
        WRITE_TO_BLOCK (count_exe,spacecnt);

{ Учет изменения длины модуля : }
        pagecnt := size_exe div 512 + 1;
        partpage := size_exe mod 512;
        pageblock := blocksize div 512;
        partblock := blocksize mod 512;
        newpagecnt := pagecnt + pageblock;
        newpartpage := partpage + partblock;

        if newpartpage > = 512 then
            begin
                newpagecnt := newpagecnt + 1;
                newpartpage := newpartpage - 512;
            end;

{ Записываем параметры в массив : }
        chint [$16] := lo(newcs);
        chint [$17] := hi(newcs);
        chint [$14] := lo(newip);
        chint [$15] := hi(newip);
        chint [4] := lo(newpagecnt);
        chint [5] := hi(newpagecnt);
        chint [2] := lo(newpartpage);
        chint [3] := hi(newpartpage);

{ ... и выводим в файл : }
        seek (f,0);

        for i := 0 to $1c do
            begin
                ch := chr(chint[i]);
                write(f,ch);
            end;

{ Заполняем межсегментное пространство : }
        seek (f,filesize(f));
        ch90h := chr($90);
        for i := 1 to spacecnt do write(f,ch90h);

{ Устанавливаем defender блок : }
        for i := 1 to (filesize(block) - 512) do
            write (f,chblock[i]);
        close(f);
        end;



begin
        MESSAGE;
        OPEN_FILE;
        if flag = 2 then
            begin
                ESC_DEFEND;
                exit;
            end;
        if flag = 3 then
            begin
                VERIFY;
                exit;
            end;
        READ_HEADER;
        PROCESS;
end.
```

Информация о новых вирусах в формате антивирусной программы AVSP 2.0

AVSP предоставляет пользователю возможность непосредственного ввода информации о новых вирусах в программу. Благодаря регулярной публикации такой информации AVSP на Вашем компьютере не отстанет от времени и создателей новых вирусов.

AVSP содержит ряд средств борьбы с неизвестными вирусами. ДИАЛОГ-МГУ обращается ко всем читателям с просьбой присылать информацию об обнаруженых новых вирусах.

Для ввода приведенной ниже информации в Вашу программу войдите в пункт меню "Данные о файлах и вирусах", подменю "Изменение информации о вирусах". Для более подробного описания можно обратиться к документации (пункт "Ввод информации о вирусах из журналов").

Имя вируса: V-330
Номер: 191.1-9524
Тип вируса: PROGRAM
Размер: 330
Шаблон: # mov ax,% / mov cx,% / mov si,% / push ax / xor byte ptr[si],%
Процедуры: *.COM
Справка: Вирус не имеет постоянного шаблона. Поиск возможен только с помощью AVSP версии 1.5 и выше. Содержит закодированную строчку : "(C)1991 1.05 DM". Лечение возможно для всех файлов, о которых запомнена информация.

Имя вируса: V-198
Номер: 210.1-40092
Тип вируса: PROGRAM
Размер: 198
Шаблон: E800005E83EE0350B4E9CD21
Процедуры: *.COM { Opt(End:-198) Mov(End:-169,Beg:0,4) Tr(End:-198)}
Справка: Только размножается.

Имя вируса: Terror
Номер: 221.1-44005
Тип вируса: PROGRAM
Шаблон: 2E8C1E410550B859ECCD213BE8753E

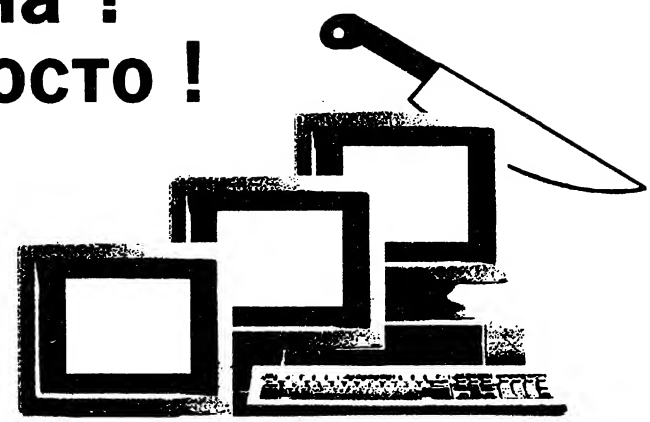
Имя вируса: Best Wishes-2
Номер: 222.1-43734
Тип вируса: PROGRAM
Размер: 970
Шаблон: 83ED03061E5051B80400
Процедуры: *.COM { Opt(End:-661) Mov(MP:-305,Beg:0,3) Tr(End:-970) }
*.EXE { Opt(St:4) Mov(MP:-300,H:22,2) Mov(MP:-298,H:20,2) TrEXE(End:-970) }
Справка: Содержит строчку "With Best Wishes".

По вопросам приобретения AVSP 2.0 обращайтесь в ДИАЛОГ-МГУ по телефонам (095) 939-38-90, 939-38-91. Подробная информация высылается бесплатно.

Цена одного комплекта AVSP 2.0 на 3 октября 1992 г. составляет 1924 рублей (с учетом 28% налога). При покупке нескольких комплектов предоставляется скидка. Скидка для дилеров - до 60%.

Копия экрана ? Это очень просто !

Н.Косарев



"Первый месяц мы держались стойко.
Мы не пили "Кока-Кола"...
Но все-таки реклама взяла свое."

И.Ильф, Е.Петров.
Одноэтажная Америка

Когда программист, прорвавшись свозь дебри вычислений,"добирается" до графики, когда на экране его персонального компьютера появляется долгожданная кривая - итог труда целой недели (месяца, квартала), наступает момент, когда этот итог необходимо засвидетельствовать,например, начальнику (для подьема его же авторитета) или жене (для оправдания потерянного времени) или другу по вечернему преферансу (для обьяснения безнадежно долгого отсутствия).

Можно, конечно, воспользоватся утилитой GRAPHICS.COM, но, как правило, хочется чего-нибудь такого...

Косо поглядывая на ящик с надписью EPSON (AMSTRAD, LASER JET), приходится искать пыльное описание оного устройства. Знакомая картина.

Действительно, задача получения копии экрана или его участка на бумаге встает достаточно часто.

Вспомним основные моменты работы матричного принтера типа EPSON в 8-игольчатом графическом режиме. Перевод принтера в графический режим осуществляется одной из команд :

- ESC "K" n1 n2 - одинарная плотность печати, 60 точек/дюйм;
- ESC "L" n1 n2 - двойная плотность печати, 120 т/д.;
- ESC "Y" n1 n2 - двойная плотность печати, 120 т/д. (ускоренно)
- ESC "Z" n1 n2 - учетверенная плотность печати, 240 т/д.

Вместо приведенных команд можно использовать команду, позволяющую устанавливать любой из указанных режимов:

- ESC "*" m n1 n2

где m - моды печати, равные 0..3 (соответственно режимы K, L, Y, Z)

После перевода принтера в графический режим каждый байт, посланный в принтер вызывает определенную реакцию игл печатающей головки. Например, байт 128 (0+0+0+0+0+0+0+128) инициирует удар самой верхней иглы, байт 1 (1+0+0+0+0+0+0+0) - нижней, а байт, равный 255 (1+2+4+8+16+32+64+128), вызывает печать вертикальной колонки из восьми точек.

Параметры n1 и n2 в командах определяют число вертикальных колонок из 8 игл, которые должен напечатать принтер. При необходимости печати 150 колонок n1=150 n2=0, при печати 340 колонок n1=chr(Lo(340)) n2=chr(Hi(340)).

Имея экран в качестве источника информации для кодирования принтера, задачу получения черно-белой "твердой" копии экрана - "печатать все, что не является фоном"- можно решить следующим способом. Все, что нужно для этого, - Turbo Pascal 5.5 или 6.0. Рассмотрим приведенный ниже модуль Screen (листинг 1).

Объект типа TWorkArea назначает границы рабочей области и проверяет правильность введенных координат (function ErrorWorkArea).

Обьект типа THardCopy работает следующим образом:

- Констрактор Init инициализирует границы рабочей области и моду печати 0..3.
- в методе ScanLine "читаются" вертикальные колонки из 8 пикселов с экрана, вычислются значения байтов для кодирования принтера и эти значения присваиваются элементам BufferArray; BufferArray хранит значения байт горизонтальной полосы экрана высотой в 8 пикселов;
- в методе Printing происходит последовательное сканирование всей назначенной для печати области экрана с одновременной подачей значений BufferArray на принтер (метод PrintLine);
- назначение интервала движения бумаги, моды печати и числа колонок печати на принтере происходит в методе SetPrinter.

Для демонстрации печати графической копии экрана представлена программа Test (листинг 2). ■

Листинг 1

```
Unit Screen;
{***********************************************}
{                                               }
{                   Demo unit                   }
{          Copyright (c) 1992 by Nik.N.Kosarev  }
{                                               }
{***********************************************}
interface
    uses
        Graph,
        Printer;
type
    TWorkArea = object
        X1,Y1,                  { координаты выводимой на печать}
        X2,Y2 : integer;        { области экрана }
        procedure Init(InitX1, InitY1, InitX2, InitY2: integer);
        function ErrorWorkArea:boolean; { проверка печатаемой области }
    end;
    THardCopy = object(TWorkArea)
        Mode : byte;
        LimUp,
        LimDn : integer;
        BufferArray : array [0..1280] of byte;
        constructor Init(InitX1, InitY1, { координаты выводимой на печать }
                         InitX2, InitY2: integer;   { области экрана }
                         InitMode:byte);            { режим печати 0..3 }
        function ScanLine(X:integer): byte;
        procedure PrintLine(CoordOne,CoordTwo:Integer);
        procedure SetPrinter(StepPrn:byte;BuffPrn:word);
        procedure Printing;
    end;
implementation
    { TWorkArea }
procedure TWorkArea.Init(InitX1, InitY1,
                         InitX2, InitY2: integer);
begin
    X1:= InitX1;
    Y1:= InitY1;
    X2:= InitX2;
    Y2:= InitY2;
end;
function TWorkArea.ErrorWorkArea : boolean;
begin
    ErrorWorkArea:= false;
    If ( X1 > = X2 ) or ( Y1 > = Y2 ) then ErrorWorkArea:= True;
end;
    { THardCopy }
constructor THardCopy.Init(InitX1, InitY1,
                           InitX2, InitY2: integer;
                           InitMode : byte);
begin
    TWorkArea.Init(InitX1, InitY1, InitX2, InitY2);
    Mode:= InitMode;
end; { Init }
function THardCopy.ScanLine(X:integer):byte;
{ Сканирует 8 вертикальных пикселов экрана и
возвращает байт, необходимый для кодирования принтера }
var
    SumCode,
    NumByte     : byte;
    k           : integer;
begin
    SumCode:= 0;
    NumByte:= 0;
    For k:= LimUp to LimDn do begin
        Inc(NumByte);                       { цвет фона = 0 }
        If GetPixel(X,k) < > 0 then begin   { если цвет пиксела на экране }
                                            { не совпадает с цветом фона }
            { PutPixel(X,k,3); }        { индикация сканирования экрана, }
                                        { если открыть скобки комментария }
            SumCode:= SumCode + (1 shl (8-NumByte));
        end; {if}
    end; {for}
    ScanLine:= SumCode;
end; { ScanLine }
procedure THardCopy.SetPrinter(StepPrn:byte; BuffPrn:word);
begin                                       { установка: }
    write(lst, #27#51+chr(StepPrn) );       { - интервала перевода бумаги }
    write(lst, #27#42+chr(Mode) );          { - режима печати "*" = #42 }
    write(lst, chr(lo(BuffPrn)) );          { - числа колонок печати }
    write(lst, chr(hi(BuffPrn)) );
end; { SetPrinter }
procedure THardCopy.PrintLine(CoordOne,CoordTwo:Integer);
var
    I:integer;
begin
    for I:= CoordOne to CoordTwo do begin
        {$I-}
        write(lst,chr(BufferArray[I]));     { посылка байта в принтер }
        {$I+}
        if (IOResult0) then begin       { контроль на выключение принтера}
            CloseGraph;
            writeln('Printer not ready');
            Halt(2);
        end; {if}
    end; {for}
end; { PrintLine }
procedure THardCopy.Printing;
var
    I: integer;
begin
    If ErrorWorkArea then Exit;
    LimUp:= Y1;
    LimDn:= Y1+7;
    Repeat
        for I:= X1 to X2 do BufferArray[I]:= ScanLine(I);{заполнение массива}
        LimUp:= LimDn+1;
        Inc(LimDn,8);
        SetPrinter(24,X2-X1+1);
        PrintLine(X1,X2);               { печать }
        write(lst,#10);                 { возврат печатающей головки }
    until LimUp  Y2;
end; { Printing }
end. { unit Screen }
```

Листинг 2. Test – демонстрационный пример вызова объекта для копии экрана

```
program Test;
uses Screen,
    Graph;
var
    GraphDriver,
    GraphMode    : Integer;
    HardCopy     : THardCopy;{ статический экземпляр объекта THardCopy }
procedure StartGraph;
    begin
        GraphDriver := Detect;
        InitGraph(GraphDriver,GraphMode,''); { не забудьте указать путь }
        If GraphResult < > grOk then begin      { к драйверу EgaVga.Bgi }
            writeln('Graphics error');
            Halt(1);
        end; { if }
end; { StartGraph }
begin { Test program }
    StartGraph;
    Rectangle(0,0,GetMaxX,GetMaxY);
    HardCopy.Init(0,0,GetMaxX,GetMaxY,1);
    HardCopy.Printing ;                     { Вызов печати копии экрана }
    repeat until keypressed;
    CloseGraph;
end. { Test program }
```

Текстовый редактор ЛЕКСИКОН

Версия 1.0

- Подготовлены комплекты для языков:
  - белорусского
  - грузинского
  - казахского
  - украинского
- Ввод и редактирование многоязычных текстов - несколько языков в одном документе
- Математические, химические и греческие символы на экране и принтере
- Подготовлен комплект для европейских языков
- Конверторы файлов в форматы других текстовых редакторов и в формат системы Xerox Ventura Publisher
- Разнообразные драйверы экранов (в том числе VGA), клавиатур и принтеров
- Обработка текстовых файлов неограниченного размера
- Возможность печати на лазерном принтере

ЛЕКСИКОН + ВЕНТУРА

СКОРОСТЬ КАЧЕСТВО УСПЕХ

Настольная издательская система Ксерокс Вентура Паблишер

Профессиональное расширение Ксерокс Вентура Паблишер

русскоязычная версия 2.1

помогут создать

- книги
- издания баз данных
- техническую документацию
- документы любой длины (более 50 страниц)
- журналы или другие издания, требующие точных переносов
- любые типы таблиц и возможность включения данных из электронных таблиц и баз данных

МИКРОИНФОРМ 113184 Москва ул. Островского д. 44

Телефон: (095) 233 0006 Телефакс: (095) 235 1053

Электронная почта: lexicon@micro.msk.su

Постижение истории

Уважаемые читатели!

В прошлом выпуске журнала мы рассмотрели игровую программу "RAILROAD TYCOON" и ненавязчиво попытались намекнуть, что игра - дело серьезное. Сегодня в развитие этой идеи ИНФОРКОМ предлагает анализ другой программы той же фирмы.

Работая над концепциями обучающего программирования, ИНФОРКОМ не смог пройти мимо новинки, выпущенной фирмой MICROPROSE. В ней, как нельзя более удачно, авторам удалось сочетать захватывающий игровой эффект и большой объем учебной информации. Такая подача материала способна не только дать основы тех или иных знаний играющему, но и заинтересовать его в дальнейшем изучении затронутых вопросов.

Мы ни в коей мере не считаем публикуемый анализ пособием по игре, заменяющим официальную фирменную инструкцию. Но это не только анализ, это и приглашение к диалогу с нашими читателями, это и импульс для продолжения начатых исследований.

И прежде, чем перейти к самой программе, мы приглашаем наших читателей принять участие в небольшом конкурсе. Ваша задача - составить список известных Вам игровых программ, имеющих обучающий эффект. Список просим создавать по форме: название программы - фирма производитель - какой обучающий эффект имеет программа.

Рассматриваются только программы для IBM-совместимых ПЭВМ.

Срок отправки писем для участия в конкурсе - до 31 декабря 1992 г. (по почтовому штемпелю). Для десяти победителей конкурса установлены призы - книга Р.Джордейна "Справочник программиста персональных компьютеров типа IBM PC,XT и AT". Пер. с англ.,М.: Финансы и статистика, 1991, 544с.

Наш адрес: 121019, Москва, Г-19, а/я 16, ИНФОРКОМ.

Программа: *CIVILIZATION*
Жанр: *SIMULATION/ STRATEGY/ MANAGEMENT/ EDUCATION*
Фирма: *MICROPROSE*
Автор: *Sid Meier*
Год выпуска: *1991*
Дисплей: *CGA/EGA/MCGA/VGA*

Общая характеристика

Программа CIVILIZATION фирмы MICROPROSE представляет собой примечательный образец многопрофильной игровой программы. Это имитатор развития цивилизации. Вам предстоит повторить путь от первых поселений начала эпохи бронзы до межзвездных перелетов будущего. Являясь прежде всего имитатором, программа ориентирована на необходимость на каждом этапе принимать ответственные решения как в вопросах военной, так и управленческой стратегии. Кроме того, программа имеет ярко выраженный обучающий эффект. Вот только краткий перечень тех дисциплин, основы которых так или иначе затрагиваются по ходу игры:

- общая история;
- история науки и техники;
- история оружия;
- социология;
- политология.

Сюжет игры

Вам предоставляется возможность управления развитием цивилизации на планете земного типа. Генерация случайной географии планеты обеспечивает уникальность каждого нового сеанса игры, а возможность задания физикогеографических параметров планеты (климат, площадь суши и т.п.) дает настолько существенные изменения в стратегии построения цивилизации, что можно говорить о нескольких принципиально по-разному сбалансированных играх в одной программе.

Игра начинается условно с 4 000 года до н.э. и продолжается примерно (в зависимости от избранного уровня сложности) до 2 000 года н.э. За этот период Вам предстоит повторить около 80 основных достижений человеческой мысли и, как финальный результат, организовать первое поселение в другой звездной системе.

Одновременно с Вами начинают свое развитие и несколько других цивилизаций. Вопрос взаимоотношения с ними Вам придется решать конкретно на всех этапах своего развития. Они могут предоставлять поддержку путем обмена научными достижениями и практическими навыками, но могут представлять и серьезную военную угрозу.

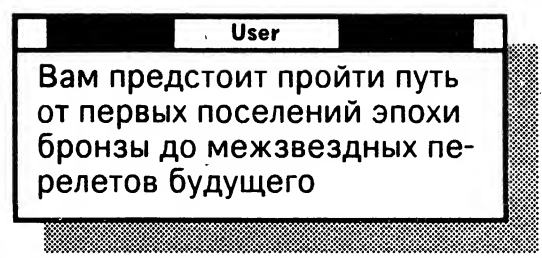
Как практически единовластный лидер своей цивилизации Вы принимаете решения и управляете следующими процессами:

—исследованием окружающего мира;
—строительством и развитием городов;
—организацией эффективного производства;
—концентрацией развития науки на основных направлениях;
—развитием дипломатических и торговых отношений с другими цивилизациями;
—созданием и развитием вооруженных сил;
—ведением боевых действий против конкурирующих цивилизаций;
—развитием политической системы своего государства.

Города

Города являются основными структурными элементами цивилизации - именно в них сосредоточены производство, торговля и наука. Потеря последнего города означает конец игры.

Одним из важнейших стратегических решений является выбор места под строительство нового города. Основой для выбора является характеристика региона и принятый план игры. Неудачный выбор места надолго сдерживает развитие государства. Слишком близкое взаиморасположение городов негативно скажется на высоких стадиях игры, а удаленное расположение затруднит развитие в начальный период и сделает неэффективной организацию их обороны. Строительство городов на берегу океана позволит хорошо развить торговлю и создать корабли, но и сделает их доступными для вражеского вторжения с моря.



Производство в городах ведется по трем основным группам:
—пища определяет развитие численности населения городов;
—продукция определяет наличие ресурсов на создание боевых частей, развитие инфраструктуры города, создание эпохальных достижений науки, техники и культуры ("Чудес света" - WONDERS);
—торговля определяет финансовое состояние государства.

Поскольку все три эти группы взаимоувязаны, любое решение, принятое при организации производства неминуемо сказывается на всех сферах развития государства.

Развитие городов

Рост и процветание городов - основная задача игры. Без этого невозможны никакие другие достижения. У Вас есть несколько способов решения этой задачи. Во-первых, Вы можете использовать накапливаемые в городе ресурсы для постройки хозяйственных объектов - зернохранилищ, казарм, акведуков, библиотек, торговых центров, банков, религиозных сооружений и т.п. Во-вторых, Вы можете создавать отряды колонистов (SETTLERS), с помощью которых проводятся ирригационные работы, добыча полезных ископаемых, строительство дорог и прочие работы. Эти же отряды являются и основой будущих городов.

Грамотное использование ландшафта, учет климатических и географических особенностей региона, а также наличия полезных ископаемых, чрезвычайно важны при этих работах. В-третьих, развитию городов, концентрации населения и смягчению социальной напряженности в обществе при повышении производительности могут способствовать некоторые из "Чудес света", которые в отличие от прочих объектов, могут существовать на планете только в единственном числе. Создание конкурирующей цивилизацией очередного "Чуда света" лишает Вас возможности создать такое же.

Развитие науки

Все многообразие стратегических концепций, положенных в основу игры, можно свести к одной: основная стратегия игры - стратегия управления развития наукой. Основное содержание игры - битва за знания.

Вам предстоит разобраться, как взаимоувязаны те десятки научных и технических достижений, которые Вам предстоит сделать, как они влияют друг на друга, что из чего проистекает. Неверная ориентация научной деятельности приведет как к сдерживанию производства, так и к снижению боевого потенциала. Если кто-то из конкурентов опережает Вас в научных достижениях, то рано или поздно он явится на Ваш континент с оружием, которому Вам нечего будет противопоставить.

Основным средством воспроизводства научных знаний является финансирование научной деятельности, но есть и другие пути его развития. На ранних стадиях игры обычное исследование окружающего мира с помощью экспедиционных отрядов может помочь в открытии древних первоисточников. Возможен обмен знаниями с сопредельными цивилизациями. Знания можно похищать с помощью дипломатов, их можно завоевывать штурмом вражеских городов.

Значительную поддержку в развитии науки оказывают библиотеки и университеты. Обычно накоплению знаний способствуют те же меры, которые приводят к развитию торговли, поскольку финансирование науки идет из налогов. Кроме того, многие из "Чудес света" также оказывают ускоряющее воздействие на науку.

Военные действия

Состояние непрерывной войны не способствует развитию Вашей цивилизации, но полностью игнорировать противника Вы не можете. На ранних стадиях игры войны ведутся в основном за материальные блага, за колонизацию уже созданных городов. На более поздних этапах Вашим конкурентам, впрочем как и Вам, может не хватать жизненного пространства для дальнейшего развития и не хватать средств для колонизации свободных, но находящихся в сложных геофизических условиях континентов. В этом случае происходит столкновение интересов и начинаются боевые действия.

С точки зрения военных расходов оборонительные действия намного эффективнее наступательных, но с другой стороны пассивный характер ведения войны дает противнику шансы на успех за счет умелого маневрирования войсками. Держать оборону континента без слабых пунктов довольно дорого. Как и всегда в этой игре, выбор за Вами. Создание войск, их рассредоточение и маневрирование - вот те вопросы, которые приходится решать ежеминутно.

Военные действия ведутся как на суше, так и на море. Корабли могут действовать как самостоятельные боевые единицы, но в то же время являются и транспортными средствами.

Практически всегда в игре Вы имеете возможность заключить мирный договор с противником. Иногда это ничего не стоит, иногда за это приходится платить основным богатством - секретами научных достижений. Не следует особо полагаться на мирные договоры, заключаемые с цивилизацией, ведущей бурную экспансию. Мирный договор никогда не останавливает потенциального агрессора. Рано или поздно он прибегнет к шантажу и вымогательству и все равно его нарушит.

Счастливым исключением являются государства с республиканской и демократической формой правления. Обычно сенат и парламент не дают их лидерам нарушать мирные договоры.

Торговля и дипломатия

Особую роль в игре занимают торговые караваны и дипломаты. Они имеют возможность свободного прохода в города соседних цивилизаций. Не являясь сами по себе значимыми боевыми единицами, они могут оказать решающее воздействие на ход Вашего развития.

Торговые караваны приносят ощутимый финансовый доход. Дипломаты дают ценную информацию. Основание дипломатического представительства в другой стране позволяет постоянно отслеживать динамику ее развития, масштабы военных приготовлений, ее взаимоотношения с третьими странами. Возможно использование дипломатов и на другой, менее свойственной им работе, а точнее говоря - в шпионской деятельности. Подстрекательство населения городов к мятежу, подкуп воинских формирований, похищение научных знаний и промышленных технологий - жертвой этих акций Вы можете быть, если пропустите в свой город дипломата.

Вопросы социологии и политологии

По мере накопления научных знаний Вам становятся доступны различные формы управления государством - от первобытной деспотии до демократии через просвещенную монархию и республиканскую форму правления. Переход от одной формы к другой выполняется посредством революций. Вы можете сами инициализировать революции подачей соответствующей команды, но иногда они происходят и без Вашего участия. На период установления новой формы правления в стране действует анархия, падает производство, свирепствует голод, снижается численность населения, сдерживается развитие науки.

Любители поспорить о том, какая форма правления лучше, могут проверить свои теоретические концепции на практике. На наш взгляд, все формы правления плохи, если они не согласованы с общим уровнем развития цивилизации - научным, техническим, промышленным, военным, социальным и культурным. А к концу игры начнут иметь вес еще и проблемы экологии. Конечно при республиканской и демократической формах правления производительность труда выше, выше и уровень доходов, но есть и существенные ограничения. Снижается эффективность Ваших управляющих воздействий, значительный вес имеет мнение сената и народа, которое не всегда совпадает с Вашим. Труднее становится удерживать порядок в городах. Если при деспотии и монархии можно избежать социальных взрывов вводом воинских частей в города, в демократическом государстве этот метод неприемлем.

На высших стадиях игры при переходе к демократии от Вас потребуется особое умение балансировать на грани социальных конфликтов.

Практические рекомендации

Для программы такого класса как CIVILIZATION не представляется возможным провести достаточно подробный анализ игровых стратегий и охватить все возможные планы игры. В качестве образца мы приводим возможный план, приводящий к победе на низких уровнях сложности. Мы будем приветствовать стратегические исследования наших читателей и с радостью опубликуем наиболее интересные планы, приводящие к успеху на высших уровнях сложности, а также не исключаем возможности существования некоторых нетривиальных планов, красивых в том смысле, в каком бывают красивы шахматные этюды.

Этап 0. Подготовка программы к работе

На этом этапе Вы можете задать физикогеографические параметры планеты. Они оказывают влияние на стратегию управления развитием цивилизации.

Например, при наличии крупных континентов и малых водных пространствах Вы имеете трудности в первоначальном развитии в связи с доступностью Ваших городов для атак других цивилизаций. С другой стороны, связь с многочисленными цивилизациями на ранних стадиях игры дает возможность более быстрого развития науки за счет обмена научными знаниями между народами.

На планете с большим количеством воды Вы можете спокойно

развивать свою цивилизацию, не отвлекаясь на военные приготовления вплоть до изобретения кораблей, но вследствие изолированности можете отстать в развитии и неожиданно пасть жертвой более мощной цивилизации из-за моря.

Дальнейшие рекомендации по плану игры приведены для средних уровней физикогеографических параметров.

Этап 1. 4 000 год до н.э. – 3 000 год до н.э.

На этом этапе основным является выбор оптимального места расположения двух-трех первых городов. Необходимо также начать исследование окружающих территорий и установить наличие конкурентных цивилизаций, захватить возможные научные первоисточники древней мудрости и открыть полезные ископаемые.

Боевые действия лучше не проводить, заключив с соседними цивилизациями мирные договоры, а для защиты от варваров ограничиться минимально достаточными силами.

Этап 2. 3 000 год до н.э. – 2 000 год до н.э.

Задача этого этапа - всемерное развитие четырех-пяти городов. Проводится ирригация обрабатываемых земель, строятся шахты, дороги, развивается инфраструктура. В соседних государствах открываются дипломатические представительства.

Этап 3. 2 000 год до н.э. – 1 000 год до н.э.

Здесь Ваша задача - техническое опережение конкурентов. Желательно максимально быстрое развитие науки. Соседние цивилизации вытесняются с континента. Это следует делать быстро и решительно. Количество городов доводится до оптимального количества, исходя из размеров континента.

Этап 4. 1 000 год до н.э. – 1 год н.э.

Основная цель - создание средств морских коммуникаций, исследование иных континентов, установление контактов с дальними цивилизациями, связь с ними через посольства, обмен знаниями, подготовка перехода к монархии и феодализму.

Этап 5. 1 год н.э. – 1500 год н.э.

Добившись технического опережения, Вы можете начинать широкую экспансию по захвату других континентов. Не следует увлекаться военными действиями. Необходимо и достаточно исполнение двух задач:

- ограниченное сдерживание (без уничтожения) самой развитой из конкурирующих цивилизаций;
- постепенное, но неуклонное развитие городов на чужих континентах.

Этап 6. 1 500 год н.э. – 1 800 год н.э.

Продолжая свое научное и промышленное развитие, Вы внимательно следите за действиями конкурентов. Основная задача - налаживание торговли между государствами. К военной силе следует прибегать только при наличии угрозы резкого взлета какой-либо цивилизации в результате свершения научного открытия и создания конкурентами нового класса боевых единиц.

Этап 7. 1 800 год н.э. – 1 900 год н.э.

На этом этапе Вам надо подготовить основы перехода к республиканской или демократической форме правления. Основная сложность - балансирование на грани социальных взрывов. Если Вы достаточно надежно сдерживали своих конкурентов в предыдущие периоды, они не смогут нанести Вам сокрушительный удар в эту эпоху социальных потрясений.

Этап 8. 1 900 год н.э. – 1 980 год н.э.

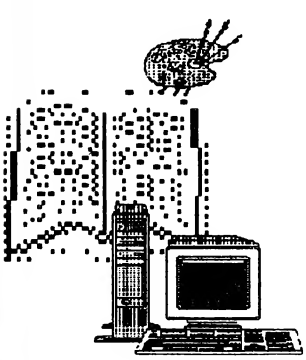
Основная задача - всемерное развитие индустрии, энергетики, электроники. Основная трудность - сдерживание загрязнения окружающей среды. Основная цель - накопление научных знаний, материальных и финансовых ресурсов к подготовке космического полета.

Этап 9. 1 980 год н.э. – 20?? год н.э.

Финальная стадия игры. Подготовка и запуск космического корабля в иные звездные системы. ■

Необходимо создать электронную энциклопедию, справочник, книгу?

Гипертекстовая система **HyperRef** *поможет Вам!*



Мы предлагаем:

– интеграцию в электронном документе текста, графики и программ;
– охрану авторских прав создателей документа;
– объем документов ограниченный только емкостью диска;
– мощный интерактивный конструктор гипертекста;
– настройку на индивидуальные информационные потребности читателя;
– встроенные средства контроля семантической близости понятий;
– "книжные закладки" и печать содержимого документа;
– гипертекстовый справочник–руководство по HyperRef.

Немного усилий, и Ваше произведение найдет своих читателей!

Телефон: (095) 238–99–33
Факс: (095) 238–32–78

Clipper или Paradox : попытка аргументированного выбора

Часть 3

С. Крапивенко

4.5. Фильтрация данных

В Paradox нет ограничений видимости записей. Для выборки данных существует мощный аппарат языка запросов.

На действие команд СУБД Clipper оказывают влияние правила видимости и текущий индекс. Определенные записи могут быть "видимы", т.е. доступны, в зависимости от фильтрующего выражения, а для записей, помеченных на удаление, - от соответствующей системной установки. Первое условие реализуется командой SET FILTER TO <логическое выражение>, второе - командой SET DELETED ON/OFF. Для "снятия" фильтра нужно указать пустое выражение: SET FILTER TO.

4.6. Установка текущего указателя

Команды и функции, работающие с текущими указателями (образом, записью и полем в Paradox, а также областью и записью в Clipper), приведены в табл. 6.

В Paradox кроме традиционного понятия "текущая запись" существует также "текущее поле" аналогично тому, как пользователь при интерактивном редактировании таблицы переходит от одного поля к другому. Кроме того, для работы с какой-либо таблицей в Paradox вместо "текущих областей", привязанных жестко к фиксированным номерам (Clipper), принято понятие "образ" таблицы. Образы упорядочены порядком открытия таблиц, т.е. таблица, открытая первой, находится в первом образе, а последней - в последнем. В режиме запросов "Ask" порядок расположения образов обратен порядку открытия - т.е. последняя открытая командой "Ask" таблица располагается вверху. Если таблица открыта несколько раз - она имеет несколько образов, отличаемых номером в порядке открытия. Например, текущее поле второго образа таблицы "My_table" - [My_table(2)->]. Образ таблицы, открытый командой "Ask", вместо номера имеет суффикс "Q" (от "Query"), например: [My_table(Q)->]. После закрытия одной из таблиц остальные "сдвигаются".

В Clipper каждая открываемая таблица должна располагаться в отдельной, свободной области. Каждая область идентифицируется номером от 1 до 254. После открытия таблицы ссылка на нее возможна либо по имени таблицы, либо по альтернативному имени, устанавливаемому при открытии, либо для таблиц, располагаемых в первых десяти областях, допустимо также обращаться по букве (от 'A' до 'J'). При установке текущей области указывается либо одна из ссылок на таблицу, либо номер области. Для перехода в первую свободную область зарезервирован номер 0. При попытке открытия таблицы в уже занятой области могут быть неприятные последствия.

Организация поиска записей в Paradox и в Clipper существенно отличается. В Paradox команда поиска одинакова как для индексированных, так и неиндексированных таблиц. При поиске по одному полю необходимо перейти в это поле, затем в команде поиска указать искомое значение. Например:

```
MOVETO [Организация] LOCATE "СОЮЗСЕРВИС"
```

При поиске по нескольким ПЕРВЫМ полям (первым в структуре таблицы) достаточно указать в команде LOCATE список значений через запятую. Результат поиска - значение системной переменной RETVAL (истина/ложь). Для поиска от текущей записи добавляется слово NEXT. Еще одна интересная возможность - поиск по неточному соответствию (например, "First" - "FIRT") - задается ключевым словом PATTERN перед значением поиска.

В Clipper две команды поиска - без использования индекса (LOCATE) и с индексом (SEEK). В первом случае условие поиска задается логическим вы-

Таблица 6. Текущий указатель

Действие	Paradox	Clipper
ТЕКУЩАЯ ОБЛАСТЬ		
Перейти в ближайшую свободную область(*1)	Не требуется	Select 0
Перейти в область с номером n (*2)	—	Select n
Перейти в область таблицы tab	MoveTo tab	Select &tab (*4)
Перейти в J-й образ таблицы TAB(*3)	MoveTo [TAB(J)->]	—
Перейти в образ запроса таблицы TAB	MoveTo [TAB(Q)->]	—
Перейти к предыдущему образу	UpImage	Select select()-1
Перейти к следующему образу	DownImage	Select select()+1
Число открытых образов (областей)	nImages ()	—
Номер области с именем tab	—	Select (tab)
Номер текущей области	ImageNo ()	Select ()
Тип текущего образа: ("Display", "Query", "None")	ImageType ()	—
ТЕКУЩАЯ ЗАПИСЬ		
Перейти к началу таблицы	Home	Goto top
Перейти к концу таблицы	End	Goto bottom
Перейти к записи с номером n	MoveTo Record n	Goto n
Передвинуться на n записей	Skip n	Skip n
Перейти в предшествующую запись (табличный вид)	Up	Skip 1
Перейти в следующую запись (табличный вид)	Down	Skip -1
Поиск записи от начала таблицы	LOCATE [NEXT] <список значений>	LOCATE <диапазон> FOR <условие>
Продолжение поиска от текущей записи	отсутствует	LOCATE NEXT
Поиск по индексному ключу	Аналогично поиску без индексного ключа	SEEK <значение>
ТЕКУЩЕЕ ПОЛЕ		
Перейти в первое поле	CTRL-HOME	—
Перейти в последнее поле	CTRL-END	—
Перейти в поле с именем fld	MoveTo Field fld	—
Перейти в поле в другом образе	MoveTo [TAB->FLD]	—
Перейти в поле слева от текущего	Left	—
Перейти в поле справа от текущего	Right	—
Перейти в поле выше (экранная форма)	Up	—
Перейти в поле ниже (экранная форма)	Down	—

(*1) Понятия "область" (Clipper) и "образ" (Paradox) близки по смыслу.
(*2) Строчные буквы идентификаторов (условно) означают допустимость употребления выражений соответствующего типа.
(*3) Прописные буквы идентификаторов (условно) означают константы.
(*4) Требуется константа, но допустима макроподстановка.

ражением, <диапазон> по умолчанию - ALL, т.е. от начала и до конца таблицы, также можно указать несколько очередных записей (NEXT <n>) или все до конца (REST). Продолжить поиск с тем же условием - команда LOCATE NEXT. Команда SEEK <значение> перемещает текущий указатель к первой записи с соответствующим значением текущего индекса, если имеется точное совпадение, иначе - к записи с близким по значению индексом, либо в конец таблицы - в зависимости от установки SET SOFTSEEK ON или OFF соответственно. Результат поиска определяется функцией FOUND () (истина/ложь).

4.7. Редактирование

Наиболее часто употребляемые команды в приложениях - это добавление новой записи, удаление текущей записи и изменение значений полей текущей записи (табл.7).

Clipper реализует эти команды без особых трудностей. Новые записи добавляются в конец таблицы. Команды удаления лишь помечают записи как удаленные (под признак удаления в каждой записи резервируется один байт), позволяя их восстанавливать командой RECALL (если все - параметр ALL); доступность или недоступность удаленных записей определяется установкой SET DELETED ON / OFF; физическое удаление осуществляется командой PACK. Порядок расположения записей в таблице определяется порядком добавления записей или результатом сортировки.

Paradox осуществляет работу с таблицами несколько сложнее. С момента начала корректировки и до ее окончания все изменения накапливаются во временной области памяти. По ходу корректировки имеется возможность отменять изменения в обратном порядке (MENU {UNDO} {YES}), запоминать текущее состояние таблиц (EDITLOG MARK) и затем восстанавли-

Таблица 7. Команды редактирования

Действие	Paradox	Clipper
Вставить новую запись перед текущей	INS	Отсутствует
Добавить новую запись в конец таблицы	END DOWN	APPEND BLANK
Удалить текущую запись	DEL	DELETE
Восстановить удаленную запись	Отсутствует	RECALL
Физически удалить записи	Отсутствует	PACK
Отменить последнее изменение	MENU {UNDO} {YES}	Отсутствует
Отменить все изменения (режим EDIT)	MENU {CANCEL} {YES}	Отсутствует
Подтвердить изменения	DO_IT!	Отсутствует
Записать в поле новое значение	[<поле>] = <значение>	REPLACE <поле> WITH <значение>

вать его (EDITLOG REVERT). Для выхода из режима корректировки требуется подтвердить изменения (DO_IT!). При монопольном доступе к таблицам (режим EDIT) возможна отмена ВСЕХ изменений (MENU {CANCEL} {YES}). В отличие от Clipper, в Paradox можно вставить запись в любое место таблицы, если нет индексов, иначе положение записи определяется индексным ключом в момент покидания записи.

4.8. Интерактивное редактирование

Без чего не обходится ни одно сколько-нибудь стоящее приложение - так это без организации интерактивной работы пользователя с таблицами. Наиболее часто используемые команды и функции приведены в табл. 8. Ниже рассмотрим некоторые из них.

Таблица 8. Команды для интерактивного редактирования

Действие	Paradox	Clipper
Переход в режим полноэкранного редактирования	WAIT <параметры>	DBEDIT (параметры) или @ .. SAY .. GET .. READ
Выбор экранной формы	PICKFORM <индекс формы>	SET FORMAT TO <файл>
Переключение вида таблица <-> форма	FORMKEY	Отсутствует
Позиционирование курсора	@ <строка>, <столбец>	@ <строка>, <столбец>
Вывод списка выражений с новой строки	? <список выражений>	? <список выражений>
Вывод списка выражений с текущей позиции	?? <список выражений>	?? <список выражений>
Ввод одной клавиши	GETCHAR ()	INKEY (<время ожидания>)
Ввод/корректировка значения переменной	ACCEPT <опции>	GET <опции>
Моделирование ввода текстовой строки	TYPEIN <строка>	KEYBOARD <строка>
Моделирование нажатия клавиши	KEYPRESS <код>	Отсутствует
Установка цвета	STYLE ATTRIBUTE <n>	SET COLOR TO <строка цвета>

4.8.1. Интерактивные средства Paradox

Вначале замечание. Как уже отмечалось (п. 4.4), Paradox поддерживает только уникальные индексные ключи. Несоблюдение или игнорирование этого требования приведет к потере записей с дублированными ключами. Поэтому интерактивную корректировку таблицы желательно вести в режиме совместного редактирования (COEDIT). При этом Paradox не позволит покинуть запись, если ее индексный ключ совпадает с ключом какой-либо другой записи: высвечивается предупреждающее сообщение и предлагаются два варианта - комбинация клавиш Alt-K копирует первичную запись в текущую, а комбинация Alt-L удаляет текущую запись.

В Paradox команда WAIT отображает на экране текущую таблицу или текущую форму (в том числе многотабличную форму) и передает управление пользователю. Полный формат команды:

WAIT <зона> PROMPT <подсказка 1>, <подсказка 2> UNTIL <список клавиш выхода>

где <зона> определяет пределы, в которых пользователь работает : текущее поле, запись или всю таблицу (соответственно FIELD, RECORD и TABLE); <подсказка 1> и <подсказка 2> - это текстовые строки, высвечиваемые в верхних двух строках экрана; <список клавиш выхода> - это наименования клавиш, при нажатии которых управление передается системе. К числу клавиш выхода необходимо также относить те, действие которых требует переопределения по сравнению со "стандартной" реакцией. В примере

WAIT TABLE PROMPT "Esc – выход, F10 – меню специальных действий, F1 – помощь", "Home/End – начало/конец таблицы, PgUp/PgDn – постраничное листание" UNTIL "Esc", "F10", "F1"

пользователь работает в пределах всей текущей таблицы до нажатия одной из трех клавиш - "Esc", "F10" или "F1". Результат - элемент списка клавиш выхода - воз-

вращается в системной переменной RETVAL.

При активировании команды WAIT экран разбивается на две части: верхние две строки - подсказки, остальное - образ рабочего пространства. Если у текущей таблицы активирована экранная форма, то форма занимает весь экран, в противном случае на экране могут быть расположены одна или несколько таблиц (образов), в том числе образ текущей таблицы.

Если не установлена какая-либо форма, внешний вид образа таблицы следующий: сверху и справа таблица обрамляется двойной рамкой, над каждым столбцом помещается его имя, а над первым столбцом - имя таблицы, первый столбец содержит номера записей (имя этого столбца - "#"). Для широких таблиц (шире экрана) действует горизонтальный скроллинг.

Экранные формы создаются дизайнером форм и могут включать либо одну или несколько страниц для полей одной записи, либо одну страницу для нескольких записей (multi record). Формы также могут быть составными (multi table), т.е. включать формы других таблиц. В формах поддерживается перенос поля на несколько строк, вычисляемые поля, а также использование переменных и функций (более 50 функций языка PAL). Назначение текущей экранной формы осуществляется оператором PICKFORM <номер формы>, где <номер формы> - "F" - для "стандартной" формы или от "F1" до "F15" для любой из 15 дополнительных форм. Переключение между табличным видом и видом в форме осуществляется командой FORMKEY (клавиша F7). Стандартная форма, как и стандартный отчет ("R") отличается от прочих тем, что используется по умолчанию (по команде FORMKEY, а для отчета - по команде InstantReport и при нажатии клавиш Alt-F7), а также тем, что формируется автоматически при ее отсутствии.

Замечание. Употребляя многотабличные формы, можно натолкнуться на досадное неудобство - после начала корректировки Paradox не позволит переключаться из формы в табличный вид и наоборот. Для обхода этой "особенности" можно предложить следующий вариант: запоминать текущую позицию (для индексированных таблиц - значения индексных полей, иначе - только номер записи), завершать корректировку, переключать вид, затем восстанавливать позицию и входить в режим корректировки.

Paradox поддерживает в команде WAIT достаточно полный набор управляющих клавиш для различных действий (приблизительно 30, всего же более 50).

Команда языка PAL ACCEPT предназначена для ввода значения в переменную. Полный формат этой команды следующий:

ACCEPT <тип> PICTURE <формат> MIN <число> MAX <число> LOOKUP <таблица> DEFAULT <значение по умолчанию> REQUIRED TO <переменная>

где <тип> определяет тип и длину переменной, PICTURE - формат отображения, MIN и MAX - соответственно максимальное и минимальное значения для контроля вводимого числа, LOOKUP - таблицу, в первом поле которой содержатся все допустимые значения переменной, DEFAULT - начальное значение, а REQUIRED требует обязательного ввода значения клавишей ENTER (а иначе можно выйти, нажав клавишу ESC). Любая из перечисленных опций не обязательна, достаточно указать имя переменной для ввода значения. Системная переменная RETVAL возвращает результат - введено ли значение (по клавише ESC возвращается FALSE). При всем богатстве опций этой команды не все опции удобны из-за сообщений на английском языке. Пример обращения:

ACCEPT "D" default Today() required to Date

Для полей таблиц Paradox поддерживает аналогичные возможности контроля путём настройки для каждого требуемого поля в режиме корректировки таблицы. Один из вариантов контроля "по словарю" допускает выход в словарь, позиционирование записи с нужным значением контролируемого поля и подстановку выбранного значения в исходную таблицу. К сожалению, этот режим сопровождается англоязычными сообщениями и поэтому в приложениях малопригоден.

В команде установки цвета STYLE ATTRIBUTE <n> параметр <n> - это число, являющееся суммой значений основного цвета и цвета фона, а при необходимости - и признака мерцания. Для цветных дисплеев основной цвет задается числом от 0 (черный) до 15 (белый), фон определяется как цвет от 0 до 7, умноженный на 16, а для "мигания" надо прибавить 128.

4.8.2. Интерактивные средства Clipper

В Clipper имеются аналогичные средства. Основная функция полноэкранного редактирования - DBEDIT. Полный формат ее следующий:

DBEDIT(n1, n2, n3, n4, <имена полей>, <функция редактирования>, <форматы>, <заголовки>, <разделители заголовков>, <разделители колонок>, <разделители сносок>, <сноски>)

Границы окна редактирования задаются первыми четырьмя параметрами соответственно: верхняя, левая, нижняя и правая границы окна. Строки нумеруются от 0 до 24, а столбцы - от 0 до 79. Имена полей открытых таблиц (или выражения) в порядке следования столбцов на экране задаются в массиве либо пустой строкой для использования всех полей текущей рабочей области (эксперименты показывают, что в версии 4.0 последний вариант проходит только для рабочих областей с номерами от 1 до 10). Следующий параметр - строка с именем определенной программистом функции, обрабатывающей нажатия клавиш, либо пустая строка для обработки по умолчанию (несколько клавиш перемещения текущего указателя и выход). Далее - массив строк или одна строка, определяющие форматирование полей. Если в параметрах

функции DBEDIT указывается массив, то каждый элемент массива предназначен для одного поля, если же переменная или константа - то для всех одинаково. Строковый массив заголовков формирует заголовки над колонками. Для переноса какого-либо заголовка на следующую строку используется знак ";" (точка с запятой). Три следующих параметра определяют символы разделения заголовков, колонок и сносок. Последний параметр позволяет аналогично заголовкам столбцов создать сноски под столбцами. Все или часть параметров могут быть пропущены.

Если указана пользовательская функция, то DBEDIT вызывает ее, передавая в качестве параметров текущий режим DBEDIT () и текущий индекс массива полей. Пользовательская функция должна проверять режим, переданный ей, и значение последней нажатой клавиши (функцией LASTKEY ()) для выбора соответствующего действия, а затем вернуть значение в функциию DBEDIT. Также возможно возвращать значение нажатой клавиши в DBEDIT командой KEYBOARD. Режимы DBEDIT : 0 - нажата клавиша перемещения курсора, 1 и 2 - попытка перехода курсора за начало и конец файла соответственно, 3 - таблица пуста, 4 - остальное. Значения, возвращаемые в DBEDIT() из пользовательской функции : 0 - закончить DBEDIT, 1 - продолжить, 2 - перерисовать экран и продолжить обработку.

Если сравнивать с Paradox, то действие функции DBEDIT более всего напоминает работу команды WAIT в табличном виде. Для редактирования с произвольным размещением полей, текста, раскраски участков экрана и некоторых других действий необходимо воспользоваться командами дисплейного ввода/вывода с последующей передачей управления пользователю командой READ. Оператор дисплейного ввода/вывода выглядит следующим образом:

@<строка>,<столбец> [SAY <выражение> [PICTURE <строка формата>]] [GET<переменная> [PICTURE <строка формата>] [RANGE <n1>, <n2>][VALID <контроль>]]

Квадратные скобки означают необязательность параметра. SAY позволяет вывести выражение на экран, GET - ввести значение в переменную памяти или поле таблицы, PICTURE специфицирует формат выражения для ввода/вывода (разрешенные символы, преобразование значения, запрещение определенных символов и т.п.), RANGE задает границы значений для ввода числа, а параметр VALID предназначен для контроля введенного значения с помощью выражения или функции логического типа. Для разработки универсальных функций контроля (в том числе поиска значения в другой таблице и т.п.) язык Clipper предоставляет такие функции, как READVAR () - возвращает имя переменной из текущего GET, и PCOUNT () - возвращает число переданных параметров. Чтобы не загромождать текст приложения, операторы дисплейного ввода/вывода могут быть записаны в отдельный текстовый файл с расширением "FMT", а для активации "формы" в нужный момент предназначен оператор SET FORMAT TO <имя файла>.

Как только экран "разрисован" приведенными выше командами, собственно редактирование начинается по команде READ. В ходе редактирования пользователь может передвигаться от одного GET поля к другому, а выход происходит в любом из трех следующих случаев : нажата клавиша Esc или PageDown, а также после ввода последнего GET поля. Функция LASTKEY () позволит Вам проконтролировать код последней нажатой клавиши.

Примечания. 1. В Clipper версии 5.0 в операторе GET добавлена возможность предварительного контроля значения (как только курсор попадает в поле). 2. Не рекомендуется редактировать поля таблиц - в ряде случаев предположительно при большом количестве открытых таблиц, индексов, фильтров, - система начинает "выделывать крендели". Надежнее использовать промежуточную переменную памяти.

Еще одна необходимая команда - установка цвета:

SET COLOR TO "<основной> [,<выделенный> [,<граница>] [, <фон>][, <невыбранный>]]]"

где основной цвет действует при выводе значений в таких командах, как @..SAY, ? и ??, <выделенный> цвет определяет активную GET переменную, а <невыбранный> цвет - неактивные GET элементы. Прочие параметры практически не используются. Каждый элемент цвета состоит из двух частей - цвета символов и цвета фона, разделяемых прямым слэшем. Цвет может быть задан числом от 0 до 7, буквами ("W" - белый, "N" - черный, "I" - инверсный) или комбинацией букв R (красный), G (зеленый), B (синий) - голубой, к примеру, - "BG", а также добавлением признака более высокой интенсивности (знака "+") и мерцания (знака "*").

Пример задания цвета: SET COLOR TO "W+/BG,W+/B,,,I/BG".

4.9. Совместный доступ к данным в локальной сети

Обе системы имеют средства совместного и монопольного доступа к таблицам, блокирования/освобождения записей и таблиц. Как видно из табл. 9, в Paradox значительно больше "сетевых" команд и функций, кроме того, надежность выше. Но самое важное отличие состоит в поддержке Paradox обновляемости буферов таблиц. Если два пользователя откроют одну и ту же таблицу и один из них будет корректировать, а второй - только наблюдать, не передвигаясь по таблице, то в Paradox последний будет видеть все изменения в соответствии с установленным периодом обновления (SetRetryPeriod <секунд>), а в Clipper - не будет. Чтобы в Clipper приложении изменения в таблице, накапливаемые предварительно в буфере, были записаны на диск, нужно выполнить команду COMMIT (сброс всех буферов) или передвинуться к

Таблица 9. Средства сетевой поддержки

Действие	Paradox	Clipper
ОПЕРАЦИИ С ТАБЛИЦАМИ		
Полное блокирование таблицы	LOCK <таблица> "FL"	Use <таблица> EXCLUSIVE
Блокирование ввода и изменений в таблице	LOCK <таблица> "WL"	FLock () (возвр..T./.F.)
Предотвращение полного блокирования таблицы	LOCK <таблица> "PFL"	-
Предотвращение блокирования ввода и изменений в таблице	LOCK <таблица> "PWL"	-
Разблокирование одной таблицы	UnLock <таблица>	UnLock
Разблокирование всех таблиц	UnLock ALL	UnLock ALL
Разблокирование списка таблиц	UnLock <список>	-
Узнать вид блокировки таблицы	LockStatus (<таблица>, "FL/WL/PWL/PFL/ANY") (возвращает 0/1)	-
Начать корректировку в монопольном режиме	EDIT <таблица> или EditKey для текущей таблицы	Use <таблица> EXCLUSIVE
Начать корректировку в режиме совместного доступа	CoEDIT <таблица> или CoEditKey для текущей таблицы	Use <таблица>
ОПЕРАЦИИ С ЗАПИСЯМИ		
Блокирование текущей записи	LockRecord или LockKey - переключатель	RLock () (возвращает.T./.F.)
Разблокирование текущей записи	UnLockRecord или LockKey - переключатель	UnLock
Узнать вид блокировки записи	RecordStatus ("NEW/LOCKED/MODIFIED /KEYVIOL") (возвращает 0/1)	-
ОПЕРАЦИИ С ЛИЧНЫМ КАТАЛОГОМ		
Перемещение списка таблиц в личный каталог	PrivTables <список>	-
Установка личного каталога	SetPrivDir <путь>	-
Узнать путь к личному каталогу	PrivDir ()	-
ОПЕРАЦИИ С РАБОЧИМ ПРОСТРАНСТВОМ		
Установка права изменений для открытых таблиц	ImageRights UPDATE	-
Запрещение изменений для открытых таблиц текущему пользователю	ImageRights READONLY	-
ОПЕРАЦИИ С БУФЕРАМИ ТАБЛИЦ		
Обновить образы открытых таблиц на экране	REFRESH	-
Установить интервал обновления буферов (в секундах)	SetRetryPeriod <n>	-
Узнать интервал обновления буферов	RetryPeriod ()	-
Сохранить буферы на диске	SaveTables	COMMIT
ПРОЧИЕ ДЕЙСТВИЯ		
Установить имя пользователя	SetUserName <имя>	-
Узнать имя пользователя	UserName ()	NetName ()
Узнать имя пользователя, заблокировавшего текущую таблицу/запись	UserName () ErrorUser ()	
Узнать тип сетевой операционной системы	NetType ()	-

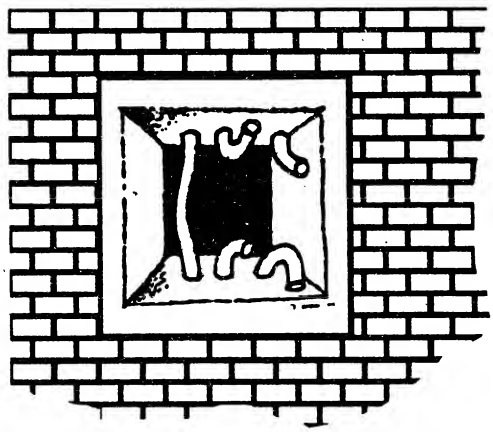
записи, не входящей в буфер. Чтение с диска в буфер происходит в момент открытия таблицы или при передвижении по таблице. Данные о размерах буферов для таблиц в Clipper отсутствуют, но предположительно - для таблицы без открытых индексов примерно 5 Кбайтов, а при открытии индексов - 2.5 Кбайта.

Clipper при совместном доступе к большим таблицам ведет себя неустойчиво. Кроме того, по словам представителей фирмы Nantucket, оптимальное количество одновременно работающих пользователей в сетевой директории - не более 4-5. Разработчикам можно порекомендовать некоторые меры предосторожности. Во-первых, для индексирования и сортировки открывать таблицы в исключительном (EXCLUSIVE) режиме, во-вторых, создавать индексы для пустых таблиц и только затем заполнять таблицы данными. Фирма Nantucket рекомендует наоборот - заполнять таблицу данными или откорректировать и затем выполнять обновление индексов командой REINDEX. Однако этот вариант требует больше времени и кажется не совсем естественным. ■

Журнал "Монитор" приглашает к сотрудничеству:

- Дилеров по распространению журнала
- Специалистов по анализу состояния компьютерного рынка
- Авторов оригинальных статей по программированию и вычислительной технике

Тел. (095)238-99-33
Факс (095)238-32-78
Адрес для писем: 117419, Москва, а/я 765.



Работа с DOS и BIOS в обход прерываний

Модуль Turbo Pascal для вирусологов и защитников информации

В. Рабец

Предположим, Вы решили защитить свою программу от копирования ключевой дискетой. Вам удалось отформатировать дискету так, что ни один из популярных копировщиков ее не берет. Вы приняли меры против трассировки программы и зашифровали exe файл, но считываете ключевую информацию через прерывание 13h. Шаловливые ручонки начинающего хакера, за 5 минут состряпав незатейливый перехватчик прерываний, сведут на нет все Ваши долгие труды.

Вы написали антивирус. Запускаете его каждый день, радуясь жизни и чистоте boot сектора. А в это время вирус, перехватив ряд прерываний и переадресовывая запросы на чтение в свободную от себя область, готовит Вашим файлам варфоломеевскую ночь.

Эти примеры показывают основные области, в которых обращение к системе без использования прерываний оказалось бы крайне полезным. Правда возможно еще одно применение излагаемого здесь материала - вирусы. Но как мне в настоящий момент кажется, вирусы на Turbo Pascal - это не совсем серьезно. Хотя, по словам д-ра Касперского, они и доставляют вирусологам некоторые неприятности своими размерами.

Предлагаемый Вашему вниманию модуль OriginOS (ORIGINal dOS & bIOS) предназначен для вызова функций прерываний 13h и 21h непосредственным обращением в DOS и BIOS. Он написан на Turbo Pascal 6.0 и рассчитан на человека, других языков не знающего. Ассемблерные вставки предельно просты и не вызовут фрустраций даже у непрограммиста, никогда ассемблера не видевшего.

Как Вы понимаете, задача непосредственного обращения к DOS и BIOS слагается из двух задач: определения адреса обработчика соответствующего прерывания и правильной передачи управления на этот адрес. OriginOS по мере сил решает обе эти задачи.

Определение адреса обработчика прерывания 13h и 21h

Определение адреса осуществляется методами, наиболее доступно изложенными Е. Касперским [1].

а) Трассировка. Пошаговое выполнение программы (трассировка) вызывается установкой в 1 флага трассировки. В этом режиме после каждой инструкции генерируется прерывание 1 (при этом часть кода от вызова прерывания до возврата из него выполняется в нормальном режиме). В нашем случае обработчик прерывания int 1 отслеживает адрес текущей команды, а для определения точки входа в int 21h (13h) устанавливается режим трассировки и вызывается какая-либо самая безобидная функция int 21h (13h). Искомой точкой входа считается первая же команда, выполненная в области DOS (BIOS).

С BIOS все просто: так как она находится в ПЗУ, адресом входа в BIOS считаем адрес входа в ПЗУ (C000:0000 - FFFF:0000). Это позволяет относительно надежно определить адрес обработчика int 13h трассировкой без каких-либо предварительных знаний.

С DOS сложнее, т.к. ее положение в ОЗУ не очевидно. Поэтому перед трассировкой находится сегмент DOS (см. ниже), и точкой входа в DOS считается первый адрес, чей сегмент сравнялся с сегментом DOS.

б) Определение адреса сегмента DOS (почти 1:1 повторяем [1]):

Подфункция 3 функции 12h прерывания 2Fh возвращает в регистре DS адрес сегмента DOS.

Недокументированные функции 34h и 52h прерывания 21h возвращают в регистрах ES:BX адреса соответственно флагу активности DOS и списку списков, расположенных в сегменте DOS. Таким образом, ES возвращает адрес сегмента DOS.

Точка входа CP/M (см. ниже) расположена в DOS.

Векторы многих прерываний DOS (20h-3Fh) направлены в сегмент DOS (сразу после инициализации системы - почти все, однако впоследствии некоторые из них могут быть перехвачены).

в) Определение точки входа в int 13h прерыванием 2F.

DOS хранит адрес, по которому она обращается к прерыванию 13h, в своих переменных (этот адрес не связан прямо с адресом того же прерывания в таблице векторов прерываний). Функция 13h прерывания 2F меняет местами значения:

а) одной из этих переменных и регистров DS:DX, б) другой из этих переменных и регистров ES:BX.

При обращении к int 13h DOS использует только переменную, обмениваемую с DS:DX (я не знаю, насколько это верно для разных версий DOS).

Передача управления на найденный адрес

Вы нашли точку входа в интересующее Вас прерывание.

В простейшем случае можно подставить найденный адрес в таблицу векторов прерываний (не обязательно в вектор данного прерывания) и вызвать измененное прерывание обычным способом (в Turbo Pascal - Intr или MsDos). Однако изменение таблицы векторов слишком легко обнаруживается.

Надежнее обратиться к обработчику прерывания непосредственно по его адресу. Иногда это легко сделать чисто паскалевскими средствами (например, вызов Int9 в [2]). Но в общем случае лучше прибегнуть к ассемблеру.

Прерывание обычно вызывается командой INT XX, где XX - номер прерывания. INT XX выполняет следующие действия:

- заносит в стек регистр FLAGS (флаги);
- запрещает трассировку и прерывания;
- заносит в стек регистр CS (кодовый сегмент);
- заносит в стек регистр IP (указатель команд);
- совершает переход на адрес обработчика, который записан по адресу 0:XX*4.

Мы можем обратиться к обработчику, последовательно выполнив аналогичные действия (шаги 3+4+5 эквивалентны команде CALL FAR - вызову процедуры):

```
PUSHF           ; занести в стек флаги
CLI             ; запретить прерывания
CALL Int_Entry  ; вызвать обработчик прерывания
```

Выход из прерывания осуществляется командой IRET (Interrupt RETurn), которая выталкивает из стека:

- регистр IP;
- регистр CS;
- регистр FLAGS (флаги), восстанавливая тем самым прежнее состояние трассировки и доступность прерываний,

после чего выполнение продолжается с адреса CS:IP.

Обратите внимание, что CALL FAR Int_Entry - не то же самое, что

```
PUSH CS             ; занести в стек
                    ; сегмент кода
PUSH IP             ; занести в стек указатель
                    ; на следующую команду
JMP FAR Int_Entry   ; перейти на Int_Entry,
```

потому что в последнем случае во время выполнения PUSH IP указатель IP, как всегда, указывает на следующую команду JMP FAR Int_Entry, и после IRET снова выполнится JMP FAR Int_Entry. Если нужно применить именно JMP, а не CALL, то следует записать:

```
PUSH CS     ; занести в стек сегмент кода (1)
PUSH OFS @Return     ; занести в стек смещение
                     ; адреса возврата (1)
JMP FAR Int_Entry  ; перейти на Int_Entry (1)
@Return:                                (1)
.....
```

тогда после выхода из прерывания CS:IP будет указывать на метку @Return.

Обращаясь таким образом к BIOS, можно спать спокойно (некоторое время - пока еще не встречаются перепрограммируемые BIOS). С DOS хуже. Вашему противнику ничто не мешает перехватить управление, внедрившись в точку входа в DOS. Это заставляет искать другие пути. Один из них - обращение к DOS в стиле CP/M [1,3].

В CP/M ее функции вызывались CALL 5. В DOS для совместимости с CP/M (в первой DOS этому придавалось большое значение) также введена возможность вызова функций близким CALL на адрес ProgramSegmentPrefix_SEG:0005. По смещению 5 от начала префикса программного сегмента записан CALL FAR 0000:00C0, а по адресу 0000:00C0 - JMP FAR на вход обработчика CP/M. Вместо CALL 5 мы можем по адресу 0000:00C1 прочитать адрес входа в DOS a la CP/M и передать туда управление, минуя множество CALL и JUMP. Однако нужно правильно заполнить стек и регистры.

Обработчик CP/M (см. [1] или [3]: листинг обработчика CP/M) предполагает, что выполнена последовательность

```
CALL 005      ; близкий CALL, в стек занесен IP
CALL FAR 0000:00C0    ; дальний CALL, в стек занесены CS
                      ; и IP=0Ah
JMP FAR CPM_Entry     ; стек не изменился
```

Обработчик в первую очередь преобразует стек к виду, нужному команде IRET: удаляет как мусор из стека IP=0Ah, извлекает из стека и запоминает CS и IP (CS в регистре AX), заполняет стек в порядке, пригодном для IRET: флаги, запомненные CS и IP.

Таким образом, мы должны занести в стек смещение адреса возврата (как во фрагменте (1)); кодовый сегмент возврата; и еще любое слово (например флаги); затем сделать JMP FAR на обработчик CP/M.

К сожалению, обработчик CP/M допускает далеко не все функции DOS. Для совместимости с CP/M номер функции передается в CL, а не в AH, и должен быть в диапазоне 024h; регистр AX портится в самом начале обработчика. Таким образом, недоступны функции с номером больше 24h или требующие параметров в регистрах AL или CL.

Разбор текста OriginOS

Интерфейсная часть включает 10 основных процедур и функций и 6 вспомогательных.

Основные процедуры и функции.

Функции, возвращающие сегмент DOS и использующие для его определения:

1) подфункцию 3 функции 12h прерывания 2Fh - DOSsegBy2F;
2) недокументированную функцию 52h прерывания 21h - DOSsegBy52;
3) недокументированную функцию 34h прерывания 21h - DOSsegBy34;
4) адрес входа CP/M - DOSsegByCPM;
5) значения векторов прерываний DOS - DOSsegByStat.

Функции, возвращающие определенный трассировкой адрес обработчика 21h и 13h:

1) Int21byTrace;
2) Int13byTrace.

Функция, определяющая вход в Int 13h посредством функции 13h прерывания 2Fh через регистры DS:DX - Int13by2F_D.

Функция, определяющая вход в int 13h посредством функции 13h прерывания 2Fh через регистры ES:BX - Int13by2F_E.

Процедура эмуляции прерывания - обращение непосредственно к обработчику прерывания по его адресу - EmulateInt.

Процедура непосредственного обращения к обработчику CP/M по его адресу - CallCPM.

Обработчики прерывания 1 при трасировке прерываний DOS и BIOS - TraceDOS и TraceBIOS.

Вспомогательные процедуры и функции:

HexWord - возвращает слово в шестнадцатиричном формате;
HexPtr - возвращает указатель в формате XXXX:XXXX;
GetVector - считывает значение вектора прерывания;
SetVector - устанавливает значение вектора прерывания;
ClearTrapFlag - сбрасывает флаг трассировки;
SetTrapFlag - устанавливает флаг трассировки;
DisableInt - запрещает прерывания;
EnableInt - разрешает прерывания.

Кроме того, в секции реализации находятся процедуры ClearTrapFlag и SetTrapFlag - сброс и установка флага трассировки, и TraceDOS и TraceBIOS - обработчики прерывания 1 при трассировке прерываний DOS и BIOS.

Переменные интерфейсной секции:

DOSseg - этому слову перед вызовом Int21byTrace должно быть присвоено значение сегмента DOS;

DOSofs - ей функция Int21byTrace присваивает значение смещения обработчика int 21h;

BIOSseg и BIOSofs - им функции Int13byTrace и Int13by2F присваивают значения сегмента и смещения обработчика Int 13h;

OrRegs (ORiginos REGisterS) - служит для передачи и приема значений регистров ЦП в/из функций EmulateInt и CallCPM. Если такое длинное имя Вам не нравится, то Вы в своей программе всегда можете написать, например,

```
var r: registers absolute OrRegs;
```

CPMentry - содержит адрес обработчика CP/M-входа в DOS. Заполняется при инициализации OriginOS {I0} (здесь и далее цифры в фигурных скобках означают номер строки листинга).

Рассмотрим сначала вспомогательные подпрограммы.

HexWord {00} и HexPtr {10} - здесь нужно лишь помнить, что данные располагаются в памяти по принципу "старший байт - по старшему адресу" и (это пригодится и в DOSsegByCPM {90}) в указателе смещение находится в младшем слове, сегмент - в старшем.

GetVector {40} и SetVector {50} - в отличие от процедур Pascal, во избежание перехвата, работают не через функции DOS 25h и 35h, а прямым чтением-записью в память. При установке вектора необходимо запретить прерывания {строки 51 и 52} - чтобы исключить вызов прерываний в момент частичного изменения вектора. При чтении вектора это ни к чему.

Рассмотрим теперь основные подпрограммы.

Тексты DOSsegBy??? просты и очевидны {60-90}.

DOSsegByStat {A0} определяет сегмент DOS по статистике векторов DOS: FirstIntN {A1} и LastIntN {A2} (...INTerrupt Number) задают диапазон рассматриваемых прерываний DOS; DosIntTested {A3} - множество принимаемых во внимание векторов: в него не входят вектора, перехваченные с большой вероятностью, а также 30h и 31h. IntStat (INTerrupt STATistics) {A4} - массив, в котором для каждого вектора указано число векторов с таким же сегментом. Изначально он зануляется {A5}. Для всех векторов диапазона {A6}, входящих в DosIntTested {A7}, ищутся векторы {A8} с таким же сегментом {A9}. При каждом совпадении элемент IntStat для данного вектора возрастает {AA}.

Вторым проходом по векторам {AC} ищется максимальный элемент IntStat {AD,AE} (он же - число совпадений сегмента), функции присваивается значение сегмента вектора с максимальным (на текущий момент) элементом IntStat {AF}. По величине EqSegNum (EQual SEGment NUMber) {AE} можно судить о достоверности результата функции. Если совпадающих сегментов нет, то EqSegNum равна 0 {AB} и значение функции не определено.

Int13by2F_D {F0}

При первом вызове прерывания {F2} DS:DX приобретают значение внутренней переменной DOS, содержащей адрес int 13h, в переменную заносится мусор (старые значения DS:DX). При втором вызове {F3} значение переменной DOS восстанавливается. Чтобы исключить между этими двумя вызовами прерывания DOS с обращением к int 13h, прерывания запрещаются {F1,F4}. Это возможно потому, что оператор Turbo Pascal Intr не использует инструкцию INT (прерывание) - так же, как и наша EmulateInt.

Дальше пойдут процедуры поинтересней.

TraceBIOS {C1} - обработчик int 1 (пошагового выполнения) во время трассировки BIOS. Т.к. эта процедура interrupt {C1}, то она должна быть дальнего типа {C0,C8}.

При каждом входе в TraceBIOS (т.е. после выполнения каждой команды трассируемого фрагмента) переменные CS и IP, соответствующие одноименным регистрам ЦП (сегменту кода и указателю команд), будут содержать адрес (CS:IP) текущей команды.

Первоочередной задачей TraceBIOS является определение момента входа в BIOS (в предположении, что это и есть точка входа в трассируемое прерывание) {C4}. Известно, что ПЗУ содержит адреса C0000FFFFF. Важно сравнивать с этим диапазоном не только CS, а полный адрес CS:IP, потому что, например, FFFF:A010 - то же, что и 0000:A000 (или 0A00:0000), и находится в ОЗУ. Как только вошли в BIOS, запоминается адрес входа {C5,C6}, и трассировка прекращается {C7}. На случай возобновления злоумышленником трассировки (а также для ускоренного прохода части кода вне BIOS) процедура выполняется только при изменении CS {C2,C3} (перед трассировкой PredCS получает заведомо нереальное значение - 0).

TraceDOS {B0} построен аналогично TraceBIOS за исключением того, что приходится анализировать не полный адрес команды, а лишь сегментный {B4}.

Programmer

IF NOT YOU=ASM_PROGRAMMER THEN REPEAT

Микропроцессор I8086 имеет ряд 16-разрядных регистров, из которых регистрами общего назначения являются AX, BX, CX и DX. С последними можно обращаться относительно свободно. Для этих четырех регистров (и ни для каких других) можно ссылаться отдельно на младший (L) или старший (H) байт, например AL или AH. Остальные регистры используются в основном для адресации. Адрес задается как СЕГМЕНТ:СМЕЩЕНИЕ, из которых физический адрес вычисляется как СЕГМЕНТ*16+СМЕЩЕНИЕ. Обычно при проходе участка программного кода или массива данных соответствующий сегмент остается постоянным, и при адресации (для экономии места) указывается лишь смещение. Из оставшихся регистров нас интересуют сегментные - CS (кода), DS (данных) и SS (стека); указатели - IP (команд), SP и BP (стека); и регистр флагов. Биты слова флагов являются определенными признаками, например, 9-й бит: 0 - прерывания запрещены, 1 - разрешены; 8-й бит - пошаговое выполнение программы (1) либо нормальное (0). В кодовом сегменте хранится программа. CS:IP всегда указывает на команду, которая должна сейчас выполниться. Данные в языке Pascal хранятся: глобальные - в сегменте данных, локальные - в сегменте стека. SS:SP всегда указывает на вершину стека. Параметры процедур в Pascal передаются через стек и адресуются через BP.

В ассемблере все операции проводятся через стек или регистры. Мы не можем, как в языках высокого уровня, переслать содержимое одной переменной прямо в другую (Var2:=Var1). Это возможно сделать либо через регистры

```
mov  AX, Var1
mov  Var2, AX
```

либо через стек:

```
push Var1
pop  Var2
```

Здесь MOV - операция пересылки содержимого правого операнда в левый, PUSH заносит операнд в стек, POP принимает из стека. Еще нам понадобятся PUSHF и POPF - засылка регистра флагов в стек и выборка; CLI (CLear Interrupt-enable flag) и STI (SeT Interrupt enable flag) - запретить и разрешить прерывания, JMP (JuMP) - переход (JMP FAR - дальний переход, т.е. с указанием нового сегмента), CALL - вызов подпрограммы (заносит в стек CS, IP и переходит на указанный адрес); а также LES - загрузка смещения в указанный регистр, а сегмента - в ES.

Оператор Turbo Pascal ASM должен сохранять регистры BP, SP, SS и DS, остальные может изменять.

UNTIL UNDERSTAND

Int13byTrace {E0}

Для начала BIOSseg и BIOSofs присваиваются нереальные значения {E1}, чтобы впоследствии можно было определить, изменились ли они при трассировке (т.е. входили ли мы вообще в BIOS). Инициализируется PredCS (предыдущий сегмент кода) {E2} (см. также {C2,C3}). Затем устанавливается обработчик int 1 {E3,E4}, устанавливается режим пошагового выполнения {E5} и вызывается безобидная функция int 13h (в данном случае - сброс дисковой системы) {E6-E7}. Теперь после каждой команды выполняется TraceBIOS. Обработчик int 1 отменит режим трассировки, как только программа войдет в BIOS. На случай, если этого не произойдет, после отработки int 13h флаг трассировки снимается {E8}. Под занавес восстанавливается обработчик int 1 {E9}, и функция получает значение {EA}.

CallCPM {H0}

Если проанализировать все функции DOS, допустимые обработчиком CP/M, то мы увидим (учитывая, что номер функции передается в CL), что на входе используются лишь регистры BX, CL, DX, DS, на выходе - AL, BX, CX, DX, DS и Flags. Баулин в [3] предлагает затереть в обработчике CP/M проверку номера функции. Если Вы на это пойдете, то ставшие доступными функции прибавляют на входе регистры DI, ES и SI, на выходе - ES. О других регистрах можно не заботиться.

Взглянув на текст CallCPM, Вы увидите, что язык ассемблера проще пареной репы: запоминается DS {H1}, заполняется нужным для CP/M обработчика образом стек {H2-H3}, из OrRegs заполняются флаги {H4,H5} и другие регистры {H6-H8} (знак & перед идентификатором заставляет рассматривать идентификатор как имя переменной, а не как зарезервированное слово ассемблера); затем переходим в обработчик {H9}. После возврата из обработчика прерывания (на метку @Return {HA}) сохраняем выходное значение DS в неиспользуемом в дальнейшем регистре DI {HB-HC}, восстанавливаем исходный DS {HD} и пересылаем регистры в OrRegs {HE-HF}.

Рассмотрим, однако, CallCPM подробнее.

Для простоты OrRegs не передается в качестве параметров процедуры, а является глобальной переменной.

Из регистров, которые Turbo Pascal требует сохранять (BP, SP, SS и DS), может измениться лишь DS. Сохраним его в стеке {H1}. Его необходимо сохранить также потому, что OrRegs - глобальная переменная, значит хранится в сегменте данных и адресуется относительно DS. Не восстановив после прерывания DS, мы не смогли бы правильно обратиться к OrRegs. Стеком в течение всей процедуры мы можем пользоваться потому, что ни одна из допустимых этим обработчиком функций DOS не портит стек (т.е. сохраняет SS и SP). Перед заполнением адресных регистров лучше запретить прерывания {H7}. DS естественно должен заполняться последним {H8}, потому что после его изменения мы теряем доступ к OrRegs. В инструкции JMP {H9} мы смогли сослаться на CPMentry, потому что BP перед этим не менялся. По выходу из функции, прежде чем пересылать регистры в OrRegs, нужно восстановить DS {HD}, а перед этим - сохранить выходное значение DS в неиспользуемом далее DI {HB-HC}.

EmulateInt {G0}

Как и в CallCPM, OrRegs не пересылается в качестве параметра процедуры, а является глобальной переменной. Неудобство небольшое, зато текст становится намного проще (сравните с одноименной процедурой из Turbo Professional [4]). EmulateInt несколько сложнее CallCPM, потому что на выходе из прерываний может оказаться измененным что угодно: одни прерывания портят стек, другие - DS и т.д. Единственный сегмент, на который можно положиться - кодовый (иначе выполнение просто не продолжилось бы с нужного места). Поэтому сделаем так: параметры стека (SS, SP) сохраним {G8,G9} в переменных, расположенных в кодовом сегменте {G1-G3}, а остальные требующие сохранения регистры (BP, DS) - в стеке {G6,G7} (так проще). По выходу из прерывания сначала восстанавливается стек (SS, SP) {GD,GE}, затем из него - DS и BP {GF,GG} (извлечение из стека - строго в порядке, обратном засылке). В кодовом сегменте нужно сохранить и IntAddr {G2-G3,G4-G5}, являющийся параметром EmulateInt {G0} (в {G5} @IntAddress+2 означает адрес на 2 байта выше, чем адрес @IntAddress). В инструкции CALL {GB} мы не можем сослаться на IntAddr, потому что BP к тому времени уже изменено {GA}. Еще одна переменная - @TmpDS {G3} - нам потребуется для временного хранения выходного значения DS {GC}. Для @IntAddress нужны 4 байта. Экономя место, @TmpDS располагаем так, что она перекрывает часть @IntAddress - одновременно эти переменные не используются.

Судя по документации, единственная возможность объявить из встроенного ассемблера идентификатор в сегменте кода - это метка (должна начинаться с @). Переменную организуем так {G1-G3}: метка, после нее заполняем нужное число байтов любым кодом (NOP - нет операции - занимает 1 байт). При ссылке на такую переменную указываем ее тип (т.е. размер), в нашем случае word() - чтобы сообщить компилятору, сколько байтов нужно обрабатывать.

Против лома нет приема

Против софта - сколько угодно. Возьмем самую, казалось бы, надежную функцию модуля - Int13byTrace. Злоумышленник может добиться интересного эффекта, если

а) найдет где-либо в BIOS команду RET;

б) вставит в цепочку int 13h свой обработчик, который, прежде чем вернуть управление, делает CALL на найденный RET.

Int13byTrace вместо адреса входа в int 13h возвратит адрес вышеупомянутой RET. Ваша программа, обращаясь по этому адресу, окажется неработоспособной, в то время как остальное программное обеспечение, взаимодействуя с системой документированным способом, будет работать нормально. В результате пользователи откажутся от Ваших услуг, навесив на Вас ярлык бракодела.

С другими функциями модуля дела обстоят еще хуже.

Не следует забывать также о существовании сетей, многозадачных сред и DR DOS, где применимы не все средства OriginOS.

Повод для оптимизма тем не менее есть. Ряд антивирусов успешно используют средства, аналогичные OriginOS. Некоторые из них в действии можно увидеть в полифаге М.Ю. Круковского и О.В. Юрковского LECAR [5].

* * *

В заключение еще несколько слов о применении OriginOS.

TraceDOS и TraceBIOS могут пригодиться для поиска входа не только в int 21h и int 13h, но и в любом другом прерывании DOS или BIOS. Например, Вы можете протрассировать int 40h и убедиться, что его адрес - F000:EC59 [6].

EmulateInt сохраняет все требуемые Turbo Pascal регистры. Как следствие - ее удобно использовать для вызова таких неаккуратных (в смысле возвращаемых регистров) прерываний, как 25h или 2E, например

```
EmulateInt ( MemL[0:$25*4] )
```

Intr в такой ситуации подвисло бы.

В планируемой к выпуску электронной версии журнала "Монитор-Диск" Вы найдете демонстрацию этого свойства EmulateInt, так же как и всех других возможностей OriginOS.

Практика показывает, что все здесь изложенное работает. Но если я в чем-то ошибся, то пусть товарищи меня сурово поправят. ■

Литература

1. **Е. Касперский. "Дыры" в MS-DOS и программы защиты информации. - Компьютер Пресс, 1991, N 10, стр. 6.**
2. **М. Остринский. Создание резидентных программ и работа с прерываниями. - Монитор, 1991, N 1, стр. 35.**
3. **А. Баулин (PEGASUS Software). Альтернативный обработчик прерывания int 21h. - СофтПанорама, 1991, N 10.**
4. **Turbo Professional 5.05. Copyright (c) TurboPower Software 1987. TPINT.ASM.**
5. **М.Ю. Круковский, О.В. Юрковский в сотрудничестве с Н.Н. Булавой, Ф.Я. Грушевским, И.Ю. Завгородним, С.М. Миненко. Полифаг LECAR. - СофтПанорама, 1992, N 5.**
6. **Без автора и названия, (c) 1990 Compact Soft. - Программистские ведомости, 1991, N 1, стр. 18.**

Листинг

```
| {$A+,B-,D+,E-,F-,I+,L+,N-,O-,R-,S-,V-}
| unit OriginOS;     { (C) V.S. Rabets, 1992. edition 17-07-92 }
|
| interface
|
| uses DOS;                  { Нужен для ссылки на Registers, Intr и MsDos }
|
| var DOSseg, DOSofs: word;        { Сегмент и смещение DOS }
| var BIOSseg,BIOSofs: word;       { Сегмент и смещение BIOS }
| var OrRegs: registers;           { Регистры CPU в EmulateInt и CallCPM }
| var CPMentry: pointer;           { Адрес входа в DOS a la CP/M }
| var PredCS: word;       { При трассировке - CS предыдущей команды }
|
| function DOSsegBy2F: word; { DOS segment by int 2F function 12h subfunc 3}
| function DOSsegBy52: word;                { by DOS function 52h }
| function DOSsegBy34: word;                { by DOS function 34h }
| function DOSsegByCPM:    word;            { from CP/M entry address }
| function DOSsegByStat (var EqSegNum: byte): word; { from statistics }
| function Int21byTrace: pointer;  { Адрес входа в int 21h, трассировкой }
| function Int13byTrace: pointer;  { Адрес входа в int 13h, трассировкой }
| function Int13by2F_D: pointer;  { То же, by int 2F func 13h, DS:DX }
| function Int13by2F_E: pointer;  { То же, by int 2F func 13h, ES:BX }
| procedure EmulateInt (IntAddr: pointer); {Вход в прерывание по его адресу}
| procedure CallCPM (CPMentry: pointer); {Вход в обработчик CP/M по его адресу}
| procedure TraceDOS (Flags,CS,IP,AX,BX,CX,DX,SI,DI,DS,ES,BP: word);
interrupt;
| procedure TraceBIOS(Flags,CS,IP,AX,BX,CX,DX,SI,DI,DS,ES,BP: word);
interrupt;
| function HexWord(W: word): string; { Вывод слова в HEX формате }
| function HexPtr (P: pointer): string; {Вывод указателя в формате XXXX:XXXX}
| procedure GetVector (Int: byte; var IntAddr: pointer);
| procedure SetVector (Int: byte; IntAddr: pointer);
| procedure ClearTrapFlag;                { Сбросить флаг трассировки }
| procedure SetTrapFlag;                  { Установить флаг трассировки }
| procedure DisableInt; inline($FA);      { CLI - запретить прерывания }
| procedure EnableInt; inline($FB);       { STI - разрешить прерывания }
|
| implementation
| var Regs: registers;             { Для всех Intr и MsDos }
        { Исходный адрес обработчика пошагового выполнения }
| var SaveInt1: pointer;
|
00 | function HexWord (W: word): string;    { Вывод слова в HEX формате }
| const HexChars: array [0..$F] of char = '0123456789ABCDEF';
| begin
|    HexWord:= HexChars[Hi(W) shr 4] + HexChars[Hi(W) and $F] +
|         HexChars[Lo(W) shr 4] + HexChars[Lo(W) and $F];
| end;
|
10 | function HexPtr (P: pointer): string; {Вывод указателя в формате XXXX:XXXX}
| var W: array [1..2] of word absolute P;
| begin
|    HexPtr:= HexWord(W[2])+':'+HexWord(W[1]);
| end;
|
20 | procedure GetVector (Int: byte; var IntAddr: pointer);
| begin
|    IntAddr:= pointer(MemL[0:Int*4]);     { Считать вектор прерывания }
| end;
|
30 | procedure SetVector (Int: byte; IntAddr: pointer);
| begin
31 | DisableInt;                              { Запретить прерывания }
|    MemL[0:Int*4]:= longint(IntAddr); { Установить вектор прерывания }
32 | EnableInt;                               { Разрешить прерывания }
| end;
|
40 | procedure ClearTrapFlag; assembler;    { Сбросить флаг трассировки }
| asm
|    pushf                    { Флаги - в стек, }
|        pop AX               { из стека - в AX }
|        and AX, $FeFF        { Сбросить флаг трассировки }
|        push AX              { Слово флагов - в стек, }
|        popf                 { из стека - в регистр флагов }
| end;
|
50 | procedure SetTrapFlag; assembler; { Установить флаг трассировки }
| asm
|        pushf                { Флаги - в стек, }
|        pop AX               { из стека - в AX }
|        or   AX, $0100       { Установить флаг трассировки }
|        push AX              { Слово флагов - в стек, }
|        popf                 { из стека - в регистр флагов }
| end;
|
60 | function DOSsegBy2F: word; { Get DOS segment by int 2F func 12h subfunc 3 }
| begin
| Regs.AX:= $1203;
| intr($2F,Regs);
| DOSsegBy2F:= Regs.DS;
| end;
|
70 | function DOSsegBy52: word; { Get DOS segment by DOS function 52h }
| begin
| Regs.AH:= $52;
| MsDos(Regs);
| DOSsegBy52:= Regs.ES;
| end;
|
80 | function DOSsegBy34: word; { Get DOS segment by DOS function 34h }
| begin
| Regs.AH:= $34;
| MsDos(Regs);
| DOSsegBy34:= Regs.ES;
| end;
|
90 | function DOSsegByCPM: word; { Get DOS segment by CP/M entry address }
| begin
| DOSsegByCPM:= MemW[0:$30*4+3]; { Должна совпадать с seg(CPMentry^) }
| end;
|
A0 | function DOSsegByStat (var EqSegNum: byte): word; { Get DOS Segment }
A1 | const FirstIntN = $20;          { 20h3F - все векторы DOS }
A2 |       LastIntN = $3F;          { из них рассматриваемые }
A3 |       DosIntTested: set of byte = [$20, $27, $2A..$2D, $32, $34..$3E];
A4 | var IntStat: array [FirstIntN..LastIntN] of byte;    { Статистика }
|    a,b: byte;                                          { Индексы циклов }
| begin
A5 |       FillChar (IntStat, SizeOf(IntStat), 0);
A6 |       for b:= FirstIntN to LastIntN-1 do  { Из всех векторов DOS }
A7 |            if b in DosIntTested then      { только рассматриваемые }
A8 |            for a:= succ(b) to LastIntN do { сравниваются друг с другом. }
A9 |       if MemW[0:b*4+2] = MemW[0:a*4+2] { Если сегменты равны, }
AA |                  then inc(IntStat[b]);      { то делаем отметку. }
AB |       EqSegNum:= 0;
AC |       for b:= FirstIntN to LastIntN-1 do       { По всему диапазону }
AD |            if EqSegNum < IntStat[b] then { ищем max элемент IntStat }
AE |            begin EqSegNum:= IntStat[b];
AF |                 DOSsegByStat:= MemW[0:b*4+2];
|              end;
| end;
|
| {$F+} { Обработчик INT 01h, должен иметь дальний тип вызовов }
B0 | procedure TraceDOS (Flags,CS,IP,AX,BX,CX,DX,SI,DI,DS,ES,BP: word);
| begin
|    if CS = PredCS then exit; { Если CS не изменился, то сразу выйти }
|    PredCS:= CS;                          { Иначе запомнить CS }
B4 |       if CS < > DOSseg then exit;     { Не дошли еще до цели }
|          DOSofs:= IP;                    { Дошли }
|          Flags:= Flags AND $FeFF; { Все уже кончено, завершить процесс }
| end;
| {$F-}
C0 | {$F+} { Обработчик INT 01h, должен иметь дальний тип вызовов }
C1 | procedure TraceBIOS (Flags,CS,IP,AX,BX,CX,DX,SI,DI,DS,ES,BP: word);
| begin
C2 | if CS = PredCS then exit; { Если CS не изменился, то сразу выйти }
C3 |       PredCS:= CS;                          { Иначе запомнить CS }
C4 |       if (CS<$C000) or (CS+IP div 16 > $FFFF) then exit;
                    { Не дошли еще до цели }
C5 |            BIOSseg:= CS;       { Дошли }
C6 |            BIOSofs:= IP;
C7 |       Flags:= Flags AND $FeFF; { Все уже кончено, завершить процесс }
| end;
C8 | {$F-}
```

```
|
D0 | function Int21byTrace: pointer;
| begin
|    DOSofs: = 0;          { Если в DOS не войдем, оно так и останется = 0 }
|    PredCS: = 0;          { Будет сравниваться с текущим сегментом кода }
|    GetVector (1,SaveInt1);
|    SetVector (1,@TraceDOS); { Установить обработчик пошагового выполнения }
|    SetTrapFlag;          { Начать трассировку }
|    Regs.AH: = $30;       { Get DOS Version }
|       MsDos(Regs);
|    ClearTrapFlag;        { Закончить трассировку }
|    SetVector (1,SaveInt1);   { Восстановить INT 01H }
|    Int21byTrace: = Ptr(DOSseg,DOSofs); { Если в DOS не входили, то DOSofs = 0 }
| end;
|
E0 | function Int13byTrace: pointer;
| begin
E1 |    BIOSseg: = 0; BIOSofs: = 0; { Если в BIOS не войдем, они так и останутся = 0 }
E2 |    PredCS: = 0;  { Будет сравниваться с текущим сегментом кода }
E3 |    GetVector (1,SaveInt1);
E4 |    SetVector (1,@TraceBIOS);
          { Установить обработчик пошагового выполнения }
E5 |    SetTrapFlag;          { Начать трассировку }
E6 |    Regs.AH: = 0;         { Reset floppy disk controller }
|       Regs.DL: = 0;
E7 |       intr($13,Regs);
E8 |    ClearTrapFlag;        { Закончить трассировку }
E9 |    SetVector (1,SaveInt1);   { Восстановить INT 01H }
EA |    Int13byTrace: = Ptr(BIOSseg,BIOSofs); { Если в BIOS не входили, то = nil }
| end;
|
F0 | function Int13by2F_D: pointer; { Адрес входа в int 13h by int 2F func 13h }
| begin                          { через регистры DS:DX }
|    Regs.AX: = $1300;
F1 |    DisableInt;          { Запретить прерывания }
F2 |    intr($2F,Regs);      { Поменяли местами DOS int 13h и DS:DX }
|    BIOSseg: = Regs.DS;     { Посмотрели DOS int 13h }
|    BIOSofs: = Regs.DX;     { ——"——"——"——"——"—— }
F3 |       intr($2F,Regs);   { Поменяли местами DOS int 13h и DS:DX }
F4 |    EnableInt;           { Разрешить прерывания }
|    Int13by2F_D: = Ptr(BIOSseg,BIOSofs);
| end;
|
G0 | function Int13by2F_E: pointer; { Адрес входа в int 13h by int 2F func 13h }
| begin                          { через регистры ES:BX }
|    Regs.AX: = $1300;
|    DisableInt;          { Not disturbing! }
|       intr($2F,Regs);   { Поменяли местами DOS int 13h и ES:BX }
|    BIOSseg: = Regs.ES;  { Посмотрели DOS int 13h }
|    BIOSofs: = Regs.BX;  { ——"——"——"——"——"—— }
|       intr($2F,Regs);   { и положили, где взяли. }
|    EnableInt;           { Можно прерывать }
|    Int13by2F_E: = Ptr(BIOSseg,BIOSofs);
| end;
|
H0 | procedure EmulateInt (IntAddr: pointer); assembler;
| asm
|    jmp @PROGRAM          { Перепрыгиваем через переменные }
|                          { Переменные в сегменте кода: }
H1 |    @SaveSS:    nop; nop  { Для хранения SS - 2 байта }
|       @SaveSP:    nop; nop  { Для хранения SP - 2 байта }
H2 |    @IntAddress: nop; nop; { 4 байта - для адреса прерывания }
H3 |    @TmpDS:     nop; nop; { Для DS (перекрывает 2 байта @IntAddress) }
|
| @PROGRAM:
|       les  SI, IntAddr          {ES:SI points to IntAddr}
H4 |       mov CS:word(@IntAddress), SI      {Save offset of IntAddr}
H5 |       mov CS:word(@IntAddress+2), ES    {Save segment}
H6 |       push BP
H7 |       push DS                   { Сохраняем BP и DS в стеке, }
H8 |       mov CS:word(@SaveSS), SS  { а параметры стека (SS,SP) - }
H9 |       mov CS:word(@SaveSP), SP  { в кодовом сегменте }
|       { Заполняем регистры CPU из OrRegs: }
|    push OrRegs.&Flags           { Слово флагов - в стек, }
|    popF                         { из стека - в регистр флагов. }
|    mov  AX, OrRegs.&AX          { Заполняем остальные регистры }
|    mov  BX, OrRegs.&BX
|    mov  CX, OrRegs.&CX
|    mov  DX, OrRegs.&DX
|    CLI                          { Запретить прерывания }
HA |    mov  BP, OrRegs.&BP
| mov  SI, OrRegs.&SI
|    mov  DI, OrRegs.&DI
|    mov  ES, OrRegs.&ES
|    mov  DS, OrRegs.&DS          { DS заполняется последним }
|
|    pushF
HB |       call dword ptr CS:@IntAddress
|       { Эмулируем int: PUSHF + CLI + CALL FAR }
|       { @IntAddress больше не нужна }
HC |    mov word(CS:@TmpDS), DS
|       {Сохраняем DS для последующей записи в OrRegs}
HD |    mov  SS, CS:word(@SaveSS)   { Восстанавливаем стек }
HE |    mov  SP, CS:word(@SaveSP)   { ——"——"——"—— }
HF |    pop  DS             { Восстанавливаем из стека исходный DS }
|       {Записываем в OrRegs значение BP на выходе из int}
|       mov  OrRegs.&BP, BP
|       {и восстанавливаем исходный, сохраненный в стеке}
HG |       pop  BP
|    mov  OrRegs.&ES, ES                  { Заполняем OrRegs }
|    mov  OrRegs.&DI, DI
|    mov  OrRegs.&SI, SI
|    mov  OrRegs.&DX, DX
|    mov  OrRegs.&CX, CX
|    mov  OrRegs.&BX, BX
|    mov  OrRegs.&AX, AX
|    { Сохраненное значение DS на выходе из int }
|    mov  AX, word(CS:@TmpDS)
|    mov  OrRegs.&DS, AX          { заносим в OrRegs }
|    pushF                        { Регистр флагов - в стек, }
|    pop  OrRegs.&Flags           {    из стека - в OrRegs. }
| end;
|
I0 | procedure CallCPM (CPMentry: pointer); assembler;
| { На входе используются CL (# функции), BX, DX, DS, DI, ES, SI }
| { на выходе - AL, BX, CX, DX, DS, ES, Flags. }
| asm
I1 | push DS                         { Сохранить DS в стеке }
I2 |       mov AX, offset @RETURN   { Взять смещение адреса возврата }
|       push AX                   { Занести в стек смещение, }
|       push CS                   { сегмент возврата }
I3 |       pushf                  { и еще 1 произвольное слово. }
I4 | push OrRegs.&Flags { Заполняем регистры ЦП: слово флагов в стек, }
I5 | popF                         { из стека - в регистр флагов; }
I6 | mov  BX, OrRegs.&BX          { остальные регистры: }
|    mov  CL, OrRegs.&CL
|    mov  DX, OrRegs.&DX
I7 | CLI                          { Запретить прерывания }
|    mov  SI, OrRegs.&SI
|    mov  DI, OrRegs.&DI
|    mov  ES, OrRegs.&ES
I8 | mov  DS, OrRegs.&DS          { DS заполняется последним }
|
I9 |    jmp dword ptr CPMentry    { На обработчик CP/M }
IA | @RETURN:          { На выходе из функции DOS стек НЕ поврежден, }
|                      { а DS возможно изменен. }
IB | push DS                     { Запоминаем выходной DS }
IC | pop  DI                     { в ненужном нам регистре DI }
ID | pop  DS              { Восстанавливаем из стека исходный DS }
|    { Заполняем OrRegs (выходной DS был запомнен в DI) }
IE | mov  OrRegs.&DS, DI
|    mov  OrRegs.&ES, ES
|    mov  OrRegs.&DX, DX
|    mov  OrRegs.&CX, CX
|    mov  OrRegs.&BX, BX
|    mov  OrRegs.&AX, AX
|    pushF          { Переносим в OrRegs флаги: регистр флагов в стек, }
IF | pop  OrRegs.&Flags       { из стека - в OrRegs. }
| end;
|
| begin
|    { Считывание адреса CP/M-входа в DOS }
J0 | CPMentry: = pointer(MemL[0:$30*4+1]);
| end.
```

Агенство "Книга–Сервис" осуществляет подписку на 1993г.

на технические, информационно-развлекательные и справочные периодические издания

(рассылаются подписчикам в конвертах, цена подписки включает стоимость изданий, услуг по почтовой рассылке и НДС)

Для оформления заказа необходимо перечислить стоимость подписки агенству "Книга-Сервис" и выслать в адрес агенства:
а) квитанцию почтового перевода (для частных лиц) или копию платежного поручения (для организаций);
б) заявку, в которой указать название изданий, количество экземпляров, почтовый индекс, адрес и Ф.И.О. подписчика.
Адрес агенства: 117168, г. Москва, Кржижановского, 14/1, агенство "Книга-Сервис". Банковские реквизиты: р/с 7467374 в Росстром-банке, корр. счет 161324 в РКЦ ГУ ЦБ РФ (г. Москва), МФО 201791.
Подписка за наличный расчет принимается непосредственно в агенстве.
Заявки, направленные после 15.12.92, и гарантийные письма не принимаются.

Наименование, издатель	Аннотация, периодичность	цена, руб,
"Русский астрологический сборник "Тамара"; автор Т.Глоба; Изд-во "Белый медведь"	*"Русский астрологический сборник "Тамара" автора Тамары Глоба, бестселлер 1991г. (тираж составил 2 000 000 экз.). Изд-во "Белый медведь" выпускает к новому году 2-е издание, переработанное и дополненное. Книга выполнена в подарочном варианте (финская мелованная бумага, уникальные иллюстрации). Тамара рассказывает о судьбе России и россиян в 1993г.; прогнозы развития событий на Земле; космическая любовь; развитие ребенка до его рождения; открытие, произошедшее на юге Урала - предтеча великих перемен; звездный мост из прошлого в будущее и многое другое.*	*199*
Журнал "Новый фермер"; СП "Периодика. Трейдинг. Сервис"	*Если вы любите землю и хотите на ней работать - это ваш журнал. Все для фермера. Информация об опыте эффективного мелкого хозяйства в США. Рекомендации отечественных экспертов и практиков. Вас ждет встреча с одним из лучших изданий России - многокрасочным, иллюстрированным, с увлекательным чтением для каждого члена семьи. (4 экз./год)*	*440*
Журнал "Мир ПК"; СП "Информейшн Компьютер Энтерпрайз"	*Аналог известного американского журнала по вычислительной технике "PC WORLD". Интересен пользователям с различным уровнем подготовки и профессионалам. Содержит обзоры аппаратных и программных средств, материалы о новых областях применения компьютеров, коммерческую информацию и многое другое. Используются оригинальные и переводные материалы. (10 экз./год)*	*990*
Журнал "PC Magazine/Russian Edition"; Изд-во "СК Ферлаг Интернешнл"	*Русскоязычный аналог американского журнала "PC Magazine". Журнал завоевал широчайшую читательскую аудиторию. Материалы, Освещающие сегодняшний и завтрашний день программного и аппаратного обеспечения. Статьи по вопросам обучения начинающих пользователей, актуальные материалы для профессионалов. Журнал ориентирован на программистов, бизнесменов и специалистов по информатике и вычислительной технике. (8 экз./год)*	*1225*
Журнал "Монитор"; НТО "Софт-Москва"	*Прекрасное высокопрофессиональное издание для программистов-практиков, ориентировано на отечественных программистов и их проблемы. (10 экз./год)*	*970*
Журнал "Сети"; СП "Информейшн Компьютер Энтерпрайз"	*Посвящен информационным компьютерным сетям и средствам телекоммуникации. Публикуются обзоры по аппаратным и программным средствам для обмена информацией на локальном и международном уровне. Издается в сотрудничестве с американским изданием "Network World". (6 экз./год)*	*435*
Журнал "Бизнес: Восток-Запад"; СП "Интерэксперт"	*Один из лучших российских журналов для деловых людей. Встречи с экономистами и предпринимателями Востока и Запада, знакомство с новыми технологиями, юридическая, финансовая, маркетинговая информация, опыт ведущих зарубежных и российских предпринимателей. Качество издания на уровне мировых стандартов. (6 экз./год)*	*628*
Бюллетень иностран. научнотехнической информации	*Впервые становится доступной для всех граждан и организаций самая оперативная и ценная информация, собираемая профессионалами информационной сети ТАСС.*	
"БИНИТИ-1" /Естественные науки, техника/	*"БИНИТИ-1" - наиболее интересные материалы о зарубежных исследованиях в областях естественных и прикладных наук, достижениях техники, о передовых производственных технологиях. (52 экз./год)*	*2945*
"БИНИТИ-2" /Сельское хозяйство. Экология/ Агенство "ИТАР-ТАСС"	*"БИНИТИ-2" - материалы о передовом зарубежном опыте в области медицины, данные медстатистики, вопросы охраны окружающей среды. (52 экз./год)*	*2944*
Информационный еженедельник "Дайджест рекламы"; Предприятие "Телер"	*Публикация всей рекламы и коммерческой информации, прошедшей по центральному телевидению и радио. Уникальная возможность для деловых людей быть в курсе всех коммерческих новостей. (50 экз./год)*	*12600*
Журнал "Moscow Magazine" (на русском языке); СП "Моксков иллюстрейтед пресс"	*Элитарный журнал, лидер совместных периодических изданий. Соответствует мировым стандартам, отвечает высоким интеллектуальным и эстетическим требованиям. Основные темы: экономика, бизнес, культура, эксклюзивные интервью крупнейших политиков, перепечатки лучших материалов западной прессы. Реклама престижных фирм. (6 экз./год)*	*730*
Журнал "Золотой век"; ТОО "Золотой век"	*Журнал литературы и искусства. Издание раритетное - ограниченное количество нумерованных экземпляров. Поиск нового художественного пространства: классика и современность. Возрождение традиций русского книгоиздания. Качество издания не соответствует мировым стандартам, а гораздо выше. (4 экз./год)*	*2985*
Журнал "Удача"; Издатель - Ф.Смирнов	*Очень интересный, иллюстрированный, престижный, информационно-развлекательный журнал о знаменитостях. Для азартных читателей, стремящихся к успеху. (6 экз./год)*	*571*

Вниманию организаций!

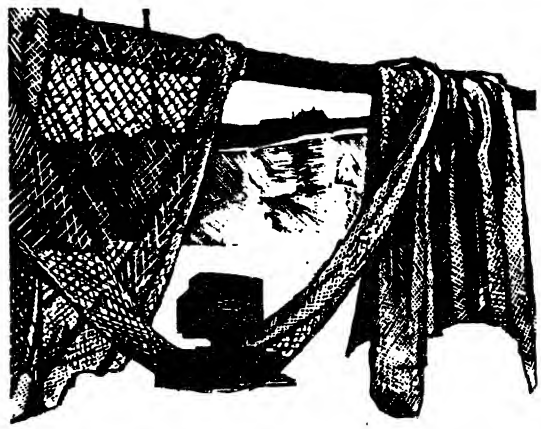
Агенство "Книга-Сервис" оказывает услуги по прведению подписных кампаний и всех видов почтовой рассылки, в т.ч. книг, рекламы, компьютерных дискет, аудио и видеокассет, др. видов продукции. Своевременное обращение в Агенство избавит вас от хлопот по реализации продукции, обеспечит авансирование деятельности, знание клиентуры и ее потребностей.

Новинка:

комплекс услуг по почтовой рассылке рекламы (direct mail). Реклама в прессе, по радио и телевидению сравнима с дробовиком, прямая почтовая реклама - это снайперская винтовка. Достоверные базы данных адресов охватывают все отрасли и регионы, обеспечивают уникальную эффективность рекламы.

Оказываем дилерские и комиссионные услуги.

Тел. 129-29-09, 129-72-12
Факс 124-94-49



Что такое IPX и как его использовать в своих целях

Часть 1

А. Ермолов

Подробнее об известном

Если Вы когда-нибудь устанавливали или хотя бы достаточно осмысленно использовали сеть NetWare (известную у наших программистов в основном как Novell - по имени потерпевшего), то знаете, что на каждой рабочей станции для использования сети надо запустить так называемую оболочку (Shell), состоящую из двух резидентных (TSR) программ, называемых IPX и NETx (где x - номер версии MS-DOS). Кроме того, можно запустить еще эмулятор всем известного сетевого протокола сеансового уровня - NetBIOS. Впрочем последнее нужно сейчас уже только для отечественных разработок, почти все фирменные давно в NetBIOS не нуждаются.

Для чего же нужны IPX и NETx? В самом грубом приближении можно сказать, что NETx передает серверу все обращения программ к удаленному диску, а IPX служит драйвером сетевого адаптера. Но это не все.

Для того чтобы понять все, что делает IPX, вспомним, как структурируются функции технического (Hardware) и программного (Software) сетевого обеспечения. В большинстве случаев при разработке сетей используется стандартная модель OSI. OSI - международный стандарт, который отражает представления Международного Института Стандартов о том, как должна осуществляться связь между компьютерными системами. В этой модели функции сетевого обеспечения разбиваются на семь групп - уровней. При этом чем выше уровень, тем более далекие от техники и близкие к пользователю функции собраны в нем. Для того чтобы описать функции каждого уровня, а заодно и объяснить, что делает IPX, позволю себе воспользоваться длинной цитатой из профессиональной документации фирмы Novell (она помещена на врезке).

Так вот, программа IPX реализует протоколы сетевого уровня (IPX) и транспортного уровня (SPX) (о чем честно и сообщает при старте).

В соответствии с моделью ISO каждый последующий уровень зависит только от одного предшествующего. Поэтому, в частности, при изменении технического обеспечения (на которое возлагаются функции первых двух уровней) надо изменять только программу IPX, но не NETx, которая участвует в реализации уровней, начиная с сеансового, и поэтому имеет право использовать только возможности IPX и SPX, но не сетевого оборудования.

Точно так же и Вы можете использовать IPX для написания своих сетевых программ, не зависящих от того, какие сетевые адаптеры будут использоваться. Более того, Ваша программа будет пригодна практически для всех сетей, так как очень редко можно встретить сетевую плату, для которой не было бы драйвера NetWare (по крайней мере 2.1x).

Почему IPX, а не NETBIOS?

Некоторые программисты в нашей стране и сейчас пишут свои приложения для сетей NetWare, но используют для этого более известный сетевой протокол - NETBIOS. Лет 5 назад это было правильно, так как стандарт, принятый фирмой Novell - протокол IPX - не был общепринятым, а стандарт NETBIOS уже тогда поддерживался всеми серьезными разработчиками сетей. Но сейчас использование NETBIOS ведет только к снижению быстродействия и к потере свободной памяти при работе в NetWare (как говорилось выше, NETBIOS загружается на рабочей станции NetWare дополнительно к сетевой оболочке). Кроме того, при использовании эмулятора NETBIOS NetWare (по крайней мере версий 2.1x) возникают неприятности при одновременном обращении к нему нескольких параллельных процессов (TSR, процессов Модулы-2). И наконец, еще один аргумент в пользу IPX. В отличие от NETBIOS, IPX используется не только на рабочих станциях сети. Вызовы IPX и SPX один к одному проходят также и на сервере и на мосте, и вообще на любом узле сети NetWare. Значит, если Вы допускаете возможность перевода впоследствии Вашего приложения в задачу (VAP или загружаемый модуль), ыполняемую на сервере, то всегда целесообразно использовать вызовы IPX и SPX.

В большинстве случаев использование NETBIOS обусловлено тем, что его протокол более известен. Далее я собираюсь изложить протокол IPX, чтобы уважаемые читатели могли пользоваться этими протоколами при программировании.

IPX. Принципы обмена информацией

В этом пункте я собираюсь в общих чертах описать протокол IPX, который подробно будет описан в дальнейшем. Как Вы увидите, протокол IPX и используемые в нем структуры имеют определенную избыточность, в некоторых случаях неудобную. Это объясняется тем, что IPX (протокол, а не программа) первоначально предназначался для несколько других сетей. Он был разработан фирмой Xerox, и затем стал применяться фирмой Novell как основной протокол системы NetWare. При этом для соблюдения преемственности было сохранено и то, что в сетях NetWare не нужно.

IPX предназначен для передачи коротких сообщений (пакетов) от одного процесса к другому. При этом для пользователя почти никогда не имеет значения, на одном компьютере выполняются эти процессы или на разных (разница только в некоторых кодах завершения).

Пакет имеет длину не более 576 байтов. Величина, прямо скажем, несколько странная. Во всяком случае, она не соответствует, насколько я знаю, длине физически пересылаемого пакета ни для одного сетевого адаптера. Для

сетей ArcNet, например, максимальная длина передаваемого за один раз физического пакета, как правило, равна 512 байтам. Поэтому максимальный "пакет" IPX на самом деле передается по сети как два физических пакета. Но пользователю до этого нет дела (если, конечно, не учитывать потери скорости).

Каждый пакет IPX начинается со служебной информации - заголовка (header), состоящего из 30 байтов при использовании протокола IPX, или 42 байтов при использовании протокола SPX. Структура заголовка IPX приведена в справочном приложении в конце статьи полными объяснениями всех полей.

Пакет, пересылаемый программой IPX, может располагаться в памяти передающей машины не обязательно непрерывно. Можно разделить его на несколько кусков (общая длина, конечно, не должна превосходить знаменитых 576 байтов). Так же и буфер приема может быть разбит на несколько частей. На практике удобно разбивать пакет на две части: в одной хранить заголовок пакета, а в другой - полезную (для Вашего приложения) информацию. Более сложные приложения могут иметь собственную служебную информацию для каждого пакета. В этом случае ее тоже может оказаться полезным хранить отдельно.

Как уже говорилось, передача в IPX происходит не от машины к машине, а от процесса к процессу. При этом на одном компьютере могут быть несколько процессов, которые одновременно ждут пакета. Для приема сообщений используются хранимые внутри IPX и недоступные программисту структуры - сокеты (гнезда). Каждому сокету присваивается уникальный номер. Для того чтобы принять сообщение, задача должна сначала открыть сокет с помощью специального запроса IPX, а затем послать IPX другой запрос - на ввод через открытый сокет. Передающая задача, в свою очередь, обязана указать в заголовке номер сокета, в который она направляет сообщение. Если принимающая сторона не успевает принимать сообщения, образуется очередь сообщений к сокету. Можно сделать и наоборот: заставить несколько процессов принимать информацию через один и тот же сокет. В этом случае мы получим очередь процессов. Правда, так делать плохо. Обычно организация очереди процессов на однопроцессорной машине приводит только к очень сильной путанице.

Как передать сообщение

Начнем подробное изучение IPX с самой простой задачи - передать строку с одного компьютера на другой и распечатать на дисплее. Более содержательные программы будут помещены в следующем номере, в продолжение этой статьи.

В настоящее время Novell поставляет для разработчиков полный программный интерфейс (API) своей системы для всех основных вариантов языка С. Однако эти библиотеки не продаются за рубли, и следовательно вряд ли доступны читателю. Поэтому будем писать программу, не пользуясь фирменными библиотеками, а используя интерфейс с IPX через программные прерывания.

Все функции IPX и SPX на рабочей станции доступны через прерывание 7Ah. Номер функции передается на регистре Bx. На других регистрах - параметры, одним из которых в ряде случаев является адрес управляющего блока IPX, который называется ECB (Event Control Block).

Предполагаем, что нам известен сетевой адрес того узла, которому адресуем посылку. Сокет принимающего процесса - 800 (по-нормальному 00 08 - припомним, что в сети сначала идет старший байт). Также будем считать, что нам известны сетевые адреса передающей и принимающей станций (для определенности - 0x02 и 0x05 соответственно).

Надо написать две программы - для передачи сообщения и для приема сообщения. Начнем с более простой - передающей. Для передачи служит функция с номером 3. На паре регистров Es:Si передается адрес ECB. Программа передачи - см. листинг 2.

Программа должна ввести строку, сформировать ECB и пакет, а затем передать его. Буфер, из которого формируется пакет, не обязан располагаться в памяти в виде непрерывного массива, а может быть разделен на некоторое количество кусков (см. справочное приложение). Я использовал это свойство и разделил буфер вывода на две части: системный заголовок IPX и массив с передаваемой информацией. Остальное ясно из текста программы.

Текст программы приема приведен в листинге 1. Она должна открыть сокет и затем принять пакет. Для ожидания приема пакета используется динамическое ожидание: программа проверяет флаг использования ECB. Прием, как и передача, осуществляется в буфер, разделенный на две части: заголовок и передаваемую строку. По окончании программа закрывает сокет.

Для открытия сокета используется функция IPX с номером 0. На регистре Dx передается номер сокета (0 - если сокет назначается динамически). На регистре Al - флаг "живучести". Если он равен 0, то сокет автоматически закрывается при завершении программы, если же - 0xFF - то сокет надо закрывать

Nota bene

- Уровень 1 - физический уровень. Этот слой имеет дело с типами аппаратуры (hardware), используемой для передачи сообщений и данных между станциями. (Световоды, витые пары, коаксиальные кабели, мультиплексоры и т.д.).
- Уровень 2 - уровень передачи данных. Этот уровень имеет дело с такими вещами, как битовые шаблоны, маркеры, обнаружение ошибок при передаче по кабелю между станциями. Например, token ring и CSMA/CD (Ethernet) - это два стандартных метода, применяемых для реализации уровня передачи данных.
- Уровень 3 - сетевой уровень. Этот уровень решает задачи адресации и доставки коммуникационных пакетов по сети. Примерами протоколов этого уровня являются XNS IDP, IPX и TCP/IP.
- Уровень 4 - транспортный уровень. Задачей транспортного уровня является обеспечение доставки информации с сетевого уровня в правильном порядке без повторений. Например, приложения NetWare имеют доступ к SPX для выполнения функций данного уровня OSI.
- Уровень 5 - сеансовый уровень. Этот уровень предназначается для создания связи между компьютерами, установки именования и адресации. В системе NetWare задачи этого уровня выполняет ответчик (NETx) или эмулятор NETBIOS.
- Уровень 6 - представительский. Этот уровень транслирует данные, передаваемые по сети во внутренние данные компьютера и обратно. Например, этот уровень переводит закодированные данные или данные с различным порядком байтов в цифровой формат машины. В NetWare этот уровень реализуется совместно ответчиком и DOS.
- Уровень 7 - прикладной. Этот уровень является интерфейсом между сетью и прикладным матобеспечением, запускаемым на компьютере. Он в NetWare также осуществляется при взаимодействии ответчика и DOS.

самому. Однако мои наблюдения показали, что лучше всегда самому закрывать сокет (автоматически он не очень закрывается). Поэтому используем честно флаг 0xFF. Код завершения возвращается на регистре AI. Для приема сообщения используется функция 4. На паре Es:Si передается адрес ECB. Наконец, для закрытия сокета используется функция 1. На регистре Dx передается номер сокета.

Обе программы (листинги 1 и 2) используют описания структур и некоторых констант из файла ipx.h, (см. листинг 3). В этом файле описаны некоторые функции, которые не объясняются (и не используются) в программах, рассмотренных в данной части статьи. Их назначение будет объяснено в продолжении статьи в следующем номере журнала.

При компиляции приведенных примеров надо учесть, что Ваш компилятор С может втихомолку вставить между полями структуры дополнительные байты. Это хорошо для ускорения работы Вашей программы, но плохо для связи с чужими программами. Используя IPX, надо блокировать такие действия. В компиляторе Microsoft C надо в командной строке компилятора указать опцию /Zp - отмена выравнивания.

Теперь можете пытаться передавать сообщение. Предупреждаю только, что я для краткости не делал проверку того, что IPX действительно загружен. Если это не так, последствия могут быть неприятные (в лучшем случае - "глухое" зависание). Поэтому я рекомендую Вам либо самостоятельно дописать соответствующую часть программы, либо включить загрузку IPX в autoexec.bat. ■

Справочное приложение

A. Пакет IPX

Каждый пакет IPX состоит из заголовка и информационной части. Заголовок (header) представляет собой структуру следующего вида:

Структура заголовка IPX		
Смещение	Содержимое	Тип
0	Checksum	BYTE [2]
2	Length	BYTE [2]
4	Transport Control	BYTE
5	Packet Type	BYTE
6	Destination Network	BYTE [4]
10	Destination Node	BYTE [6]
16	Destination Socket	BYTE [2]
18	Source Network	BYTE [4]
22	Source Node	BYTE [6]
28	Source Socket	BYTE [2]
Все поля high-low, то есть наиболее значимый байт поля - первый. Это не соответствует формату процессоров Intel, для которых наиболее значимый последний (low-high).		

Поля имеют следующий смысл:

Checksum (контрольная сумма)

Это поле включено для соответствия оригинальному определению заголовка пакета Ксерокса. IPX всегда устанавливает это поле в 0xFFFF. Поскольку сетевые карты сами подсчитывают контрольную сумму пакета, оно никому не нужно.

Length (длина)

Это поле содержит полную длину сетевого пакета, которая равна длине заголовка плюс длина данных. Минимальная длина пакета - 30 байтов (заголовок), максимальная - 576 байтов (заголовок плюс данные). Поле заполняет IPX.

Transport Control (управляющее слово моста)

Это поле используется межсетевыми мостами. IPX всегда устанавливает его в 0 перед посылкой.

Packet Type (тип пакета)

Это поле показывает, какой тип обслуживания предполагает или запрашивает пакет. В протоколе фирмы Xerox определены следующие типы:

0 - Unknown Packet Type (неизвестный тип)
1 - Routing Information Packet (Информация маршрутизации)
2 - Echo Packet (эхо)
3 - Error Packet (пакет обработчика ошибок)
4 - Packet Exchange Packet (пакет пакетного обменника)
5 - Sequenced Packet Protocol Packet (пакет протокола SPX)
16-31 - Экспериментальные протоколы
17 - NetWare Core Protocol (протокол ядра NetWare)

Пользователи IPX должны устанавливать Packet Type 0 или 4. SPX устанавливает 5 для пакетов его протокола.

Так написано в документации фирмы Novell. Мне однако не удалось получить каких-либо особенностей и при использовании других типов. Кажется, имеют смысл только два типа: 5 (необходим при использовании SPX) и 17.

Derstantion Network (сеть адресата)

Это поле содержит номер сети, которой принадлежит адресуемый узел. В NetWare каждый локальный участок (Network) глобальной сети (Internetwork) получает от администратора сети уникальный 4-байтовый номер.

Если это поле равно 0, то NetWare подразумевает, что адресуемый узел находится на том же локальном участке сети, что и передающий, и информация не передается через мост.

Destination Node (узел адресата)

Это поле содержит физический адрес узла, которому передается пакет. Не все типы локальных сетей имеют одинаковую длину адресного поля. Ethernet использует для адресации шесть байтов, а Omninet - один. Если адрес занимает меньше, чем 6 байтов, то он размещается в последних байтах, а поле дополняется нулями.

Адрес узла FF FF FF FF FF FFh (6 байтов, содержащих FF) служит для широковещательной (broadcast) передачи пакета всем узлам адресуемой локальной сети.

Destination Socket (сокет адресата)

Это поле содержит сокет процесса, которому адресовано сообщение.

Фирма Novell поддерживает список сокетов, хорошо известных (well-known) во всем окружении NetWare. Разработчики программного обеспечения, которые пишут дополнительные (value-added) модули NetWare, могут получить назначение своих сокетов у фирмы Novell. Номера динамически распределяемых сокетов начинаются с 4000h (это отличие от стандарта Xerox). Номера well-known сокетов назначаются, начиная с 8000h. Сокеты с номерами, превышающими 8000h, не должны использоваться, если они не зарегистрированы для приложений в фирме Novell.

Source Network (исходная сеть)

Поле содержит номер локальной сети, которой принадлежит исходный узел.

Поле может содержать ноль. Это обозначает, что физическая локальная сеть, к которой присоединен источник, неизвестна.

Source Node (узел источника)

IPX записывает в это поле физический адрес узла источника.

Source Socket (сокет источника)

IPX записывает в это поле номер сокета процесса, передавшего пакет. Процессы не должны использовать для приема и передачи один и тот же сокет.

К сожалению, я не нашел нигде определения сокета, используемого для передачи. Мои наблюдения показали, что, по-видимому, это - сокет, в котором последний раз ждал приема передающий процесс. Такая неопределенность делает довольно сложным выполнение требования несовпадения сокета приема и передачи. Я, разрабатывая свой транспортный протокол, обошел это требование тем, что для посылки всегда использую фиксированный номер сокета, а для приема подтверждения - динамически выделяемый. При использовании SPX проблем с "наложением" сокетов, по-видимому, не возникает.

Б. Управляющий блок IPX

Для того чтобы послать или принять пакет, Ваша программа должна сформировать управляющую структуру, называемую ECB (Event Control Block). ECB состоит из двух частей: 36-ти байтная порция фиксированной длины и порция данных, длина которой зависит от количества буферов, ассоциированных с ECB. Ниже приведено изображение ECB и описание его полей.

Структура ECB (ECB Structure)			
Смещение	Содержимое	Тип	Порядок
0	Link Address	BYTE [4]	offset-segment
4	ESR Address	BYTE [4]	offset-segment
8	In Use Flag	BYTE	
9	Completion Code	BYTE	
10	Socket Number	WORD	high-low
12	IPX Workspace	BYTE [4]	
16	Driver Workspace	BYTE [12]	
28	Immediate Address	BYTE [6]	
34	Fragment Count	WORD	low-high
38	Fragment Address 1	BYTE [4]	offset-segment
40	Fragment Size 1	WORD	
42	Fragment Address 2	BYTE [4]	offset-segment
46	Fragment Size 2	WORD	

Link Address (поле связи)

IPX использует это поле во время работы с ECB. В общем случае это поле используется приложениями как поле связки в списке свободных ECB.

ESR Address (адрес ESR)

Это поле содержит адрес входной точки, определенной в приложении программы обработки события (Event Service Routine - ESR), которая вызывается IPX при завершении посылки или приема сообщения. Поскольку IPX устанавливает поля In Use Flag и Completion Code, приложение может просто периодически опрашивать эти поля для того, чтобы уловить конец операции. Если ESR не используется, поле ESR Address равно нулю.

In Use Flag (флаг использования)

Поле содержит ненулевое значение, когда IPX использует данный ECB. При получении ответа IPX пишет сюда ноль. Возможные ненулевые значения:

FBh Событие произошло, но ECB стоит в очереди на обслуживание.

FDh Событие обслуживается, а IPX ждет истечения временного интервала.

FEh IPX ждет, пока в сокет придет пакет.

FFh ECB используется для посылки пакета.

FAh ECB обрабатывается IPX.

F8h Произошла попытка передачи в то время, когда IPX был занят. Пакет и ECB направлены в очередь для последующей обработки.

По приходе ответа IPX сбрасывает это поле в 0.

Completion Code (код завершения)

IPX использует это поле для индикации результата посылки или приема. Его не надо использовать, пока In Use Flag не установлен в ноль. Возможные значения будут указаны при описании функций IPX.

Socket Number (номер сокета)

Это поле содержит номер сокета, с которым ассоциирован ECB. При передаче поле содержит номер сокета-адресата, при приеме - сокета, через который будет осуществляться прием.

IPX Workspace (рабочая область IPX)

Это 4-байтовое поле резервируется для IPX. Оно не должно инициализироваться и изменяться во время использования ECB IPX. В остальное время может использоваться приложениями.

Driver Workspace (рабочая область драйвера)

Это 12-байтовое поле резервируется для сетевого драйвера. Оно не должно инициализироваться и изменяться во время использования ECB IPX. В остальное время может использоваться приложениями.

Immediate Address (непосредственный адрес)

Это поле содержит физический адрес узла адресата или послявшего пакет узла. В случае когда пакет посылается в другую локальную сеть - адрес моста.

Перед посылкой пакета при помощи IPX приложение обязано заполнить это поле. SPX само заполняет его при установлении соединения.

Fragment Count (количество фрагментов – буферов)

Это поле содержит количество фрагментов, из которых состоит пакет (при посылке) или в которые он будет помещен (при приеме). Его значение должно быть больше нуля. Значение 1 указывает, что прием или передача производятся с использованием одного буфера.

Fragment Descriptor (описатель фрагмента)

Fragment Descriptor идентифицирует буфер, из которого берется или в который будет помещен кусок пакета. ECB IPX обязано иметь по крайней мере один описатель фрагмента (соответствующий заголовку пакета). Описатели фрагментов образуют список описателей фрагментов (Fragment Descriptor List).

Fragment Descriptor содержит два поля:

Address содержит адрес буфера приема/передачи.

Size содержит величину буфера.

Первый буфер, описанный в Fragment Descriptor List, должен быть не меньше 30 байтов для IPX и 42 байтов для SPX. Он должен вмещать полный заголовок пакета. Полная длина пакета (сумма величин фрагментов) не должна превосходить 576. ■

Листинг 1. Программа приема сообщения

```
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <e:\article\progs\ipx.h>
#define TargetSocket 0x08
/*
      Программа приема и распечатки строки символов
*/
void main () {
      unsigned char RecieveString [80];  // Принимаемая строка
      IPXHeader Header;
      ECB_2 My_ECB;
      ECB_2 _far *My_ECB_ptr;
      union REGS inregs, outregs;
      struct SREGS segreg;
      /* Заполнение ECB */
      My_ECB.ESRAddress = NULL;
      My_ECB.SocketNumber = TargetSocket;
      My_ECB.FragmentCount = 2;
      My_ECB.FragmentAddress_1 = &Header;
      My_ECB.FragmentSize_1 = header_length;
      My_ECB.FragmentAddress_2 = &RecieveString;
      My_ECB.FragmentSize_2 = sizeof RecieveString;
      /* Открытие сокета */
      inregs.x.bx = IPX_open;
      inregs.h.al = 0xFF;
      inregs.x.dx = TargetSocket;
      int86 (IPX_interrupt, &inregs, &outregs);
      /* Выдача кода завершения */
      if (outregs.h.al != 0x00)
            printf ("Ошибка открытия сокета. Код завершения %x \n",
                  outregs.h.al);
      /* Прием сообщения */
      inregs.x.bx = IPX_listen;
      My_ECB_ptr = &My_ECB;
      inregs.x.si = FP_OFF ( My_ECB_ptr );
      segreg.es = FP_SEG ( My_ECB_ptr );
      int86x (IPX_interrupt, &inregs, &outregs, &segreg);
      while (My_ECB.InUseFlag)    // Ожидание завершения приема
            {};
      /* Распечатка строки или кода завершения */
      if (My_ECB.CompletionCode != 0)
            printf ("Ошибка приема сообщения. Код завершения %x \n",
                  My_ECB.CompletionCode);
      else
            printf ("%s \n", RecieveString);
      /* Закроем сокет */
      inregs.x.bx = IPX_close;
      inregs.x.dx = TargetSocket;
      int86 (IPX_interrupt, &inregs, &outregs);
}
```

Листинг 2. Программа передачи сообщения

```
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <e:\article\progs\ipx.h>
#define TargetSocket 0x08
#define TargetNode 0x02
/*
Программа передачи введенной оператором строки посредством IPX
*/
void main () {
      unsigned char DispString [80];  // Строка, которую вводим с дисплея
      int Length; // Длина введенной строки
      IPXHeader Header;
      ECB_2 My_ECB;
      ECB_2 _far *My_ECB_ptr;
      union REGS inregs, outregs;
      struct SREGS segreg;
      /* Ввод строки */
      printf ("Введите, пожалуйста, передаваемое сообщение \n");
      scanf ("%s", DispString);
      Length = strlen (DispString);
      /* Заполнение ECB */
      My_ECB.ESRAddress = NULL;
      My_ECB.SocketNumber = TargetSocket;
      strnset (My_ECB.ImmediateAddress, 0x00,
            sizeof My_ECB.ImmediateAddress);
      My_ECB.ImmediateAddress[5] = TargetNode;
      My_ECB.FragmentCount = 2;
      My_ECB.FragmentAddress_1 = &Header;
      My_ECB.FragmentSize_1 = header_length;
      My_ECB.FragmentAddress_2 = &DispString;
      My_ECB.FragmentSize_2 = Length+1;
      /* Заполнение заголовка сообщения */
      Header.PacketType = 0x00;
      strnset (Header.DestantionNetwork, 0x00,
            sizeof Header.DestantionNetwork);
      strnset (Header.DestantionNode, 0x00,
            sizeof Header.DestantionNode);
      Header.DestantionNode [5] = TargetNode;
      Header.DestantionSocket = TargetSocket;
      /* Передача сообщения */
      inregs.x.bx = IPX_send;
      My_ECB_ptr = &My_ECB;
      inregs.x.si = FP_OFF ( My_ECB_ptr );
      segreg.es = FP_SEG ( My_ECB_ptr );
      int86x (IPX_interrupt, &inregs, &outregs, &segreg);
      while (My_ECB.InUseFlag)    // Ожидание завершения посылки
      {};
      /* и выдача кода завершения */
      printf ("Сообщение передано; код завершения %x \n",
            My_ECB.CompletionCode);
}
```

Листинг 3. Константы и структуры IPX

```
/*
      Константы и структуры IPX
*/
#define IPX_interrupt 0x7A  // Программное прерывание IPX
#define header_length 30    // Длина заголовка пакета
/* Функции IPX */
#define IPX_open 0x00
#define IPX_close 0x01
#define IPX_localTarget 0x02
#define IPX_send 0x03
#define IPX_listen 0x04
#define IPX_event 0x05
#define IPX_cancel 0x06
#define IPX_specialEvent 0x07
#define IPX_interval 0x08
#define IPX_GetInternetwork 0x09
#define IPX_control 0x0A
#define IPX_disconnect 0x0B

/* Заголовок пакета */
typedef struct{
      unsigned int Checksum;
      unsigned int Length;
      unsigned char TransportControl;
      unsigned char PacketType;
      unsigned char DestantionNetwork [4];
      unsigned char DestantionNode [6];
      unsigned int DestantionSocket;
      unsigned char SourceNetwork [4];
      unsigned char SourceNode [6];
      unsigned int SourceSocket;
      } IPXHeader;
/* Управляющий блок с одним фрагментом данных */
typedef struct {
      void _far *LinkAddress;
      void _far *ESRAddress;
      unsigned char InUseFlag;
      unsigned char CompletionCode;
      unsigned int SocketNumber;
      unsigned char IPXWorkspace [4];
      unsigned char DriverWorkspace [12];
      unsigned char ImmediateAddress [6];
      unsigned int FragmentCount;
      void _far *FragmentAddress;
      unsigned int FragmentSize;
      } ECB_1;
/* Управляющий блок с двумя фрагментами данных */
typedef struct {
      void _far *LinkAddress;
      void _far *ESRAddress;
      unsigned char InUseFlag;
      unsigned char CompletionCode;
      unsigned int SocketNumber;
      unsigned char IPXWorkspace [4];
      unsigned char DriverWorkspace [12];
      unsigned char ImmediateAddress [6];
      unsigned int FragmentCount;
      void _far *FragmentAddress_1;
      unsigned int FragmentSize_1;
      void _far *FragmentAddress_2;
      unsigned int FragmentSize_2;
      } ECB_2;

/* Управляющий блок службы времени */
typedef struct {
      void _far *LinkAddress;
      void _far *ESRAddress;
      unsigned char InUseFlag;
      unsigned char WorkSpace [5];
      } SpecialECB;

/* Сетевой адрес плюс непосредственный адрес узла */
typedef struct {
      unsigned char NetworkNumber [4];      // Номер локального участка
      unsigned char NodeAddress [6];        // Сетевой адрес
      unsigned int Socket;                  // Номер сокета
      unsigned char ImmediateAddress [6];   // Непосредственный адрес
      } IntAddr;
```

SYMANTEC

The software side of life

Десять лет истории,
Десять сотен сотрудников,
Десять миллионов пользователей,
И одна идея – Создавать лучшие программы в мире.
Symantec

WHITEWATER RESOURCE TOOLKIT

Вы ищите наиболее современное средство для работы с Windows™? Такое, которое позволит Вам легко управлять и манипулировать ресурсами? Средство, придающее привлекательный и неповторимый облик интерфейсу Вашей программы, И средство, популярное у сотен тысяч разработчиков?

Torga Whitewater Resource Toolkit для Bac! Whitewater Resource Toolkit - экранный редактор для среды Windows™, открывающий новые заманчивые возможности для дизайна интерфейсов программ "под Windows™.

ObjectGraphics

Если же Вы хотели бы найти более легкий способ создавать графические прикладные программы для Microsoft® Windows™, то Вам необходим ObjectGraphics. ObjectGraphics - высокоуровневая, полностью объектно-ориентированная графическая библиотека, придающая дополнительные возможности ObjectWindows, она позволит создать поистине графический интерфейс для Ваших программ.

ObjectGraphics - инструмент для пользователей компиляторов Borland® C++, Turbo Pascal ® for Windows™ или Actor®

За подробной информацией о приобретении программных продуктов Symantec обращайтесь к

115409, Москворечье, 31/2
АО "Диалог-МИФИ",
тел 320-3466, факс 324-3055

121165. Москва,
Трио Плюс, Корп.
тел. 281-9890

109044, Крутицкий вал,
3, корп. 2
Merisel-КТ 276-3170, факс 274-0097

123022, Москва
Звенигородское шоссе, 9
"Перспективные технологии"
тел. 256-6271

Microsoft - зарегистрированная торговая марка компании Microsoft Windows - зарегистрированная торговая марка компании Microsoft Borland - зарегистрированная торговая марка компании Borland International. Borland C++, Turbo Pascal for Windows, ObjectWindows - зарегистрированные торговые марки компании Borland International Actor, ObjectGraphics, Whitewater Resource Toolkit - зарегистрированные торговые марки компании Symantec Corporation.

С. Курковский

Алгоритмы сортировки

Наверное, не родился еще тот программист, который ни разу в своей работе не сталкивался с необходимостью рассортировывать данные, иначе говоря, упорядочивать их определенным образом. Сортировка применяется во всех без исключения областях программистской деятельности, будь то базы данных или математические программы. Многие программисты пользуются специально предназначенными для этого библиотечными функциями (например, qsort() в Турбо Си), особо не задумываясь над тем, а как же на самом деле работают эти функции. В действительности существует множество различных методов сортировки данных, которые подробнейшим образом описаны в классической работе [1]. Причем для разных типов данных иногда целесообразно применять разные методы сортировки. Но тем не менее практически каждый алгоритм сортировки можно разбить на три части [1]:

– сравнение, определяющее упорядоченность пары элементов;

– перестановку, меняющую местами пару элементов;

– собственно сортирующий алгоритм, который осуществляет сравнение и перестановку элементов до тех пор, пока все элементы множества не будут упорядочены.

Подобными свойствами обладают и те пять алгоритмов сортировки, которые рассмотрены ниже. Они отобраны из множества алгоритмов, потому что, во-первых, наиболее часто используются, а во-вторых, потому что большинство остальных алгоритмов является различными модификациями описанных здесь. Возможно некоторые из этих пяти алгоритмов Вам знакомы, тогда Вы сможете сравнить свои разработки с приведенными в этой работе, другие алгоритмы могут оказаться для Вас новыми. Эти алгоритмы отличаются по своему быстродействию (рис. 1, 1 tick = 1/18.2 сек., приведенные результаты соответствуют компьютеру AT 286 - 12 MHz), но все они представляют некоторый интерес для изучения. Все программные разработки выполнены на языке Си, но естественно нет никаких ограничений для того, чтобы реализовать все описанные алгоритмы на каком-либо другом языке.

Метод пузырька (листинг 1)

(В работе [1] этот метод назван также обменной сортировкой с выбором). Нетрудно догадаться, что идея этого метода отражена в его названии. Самые легкие элементы массива "всплывают" наверх, самые "тяжелые" - тонут. Алгоритмически это можно реализовать следующим образом. Мы будем просматривать весь массив "снизу вверх" и менять стоящие рядом элементы в том случае, если "ниж-

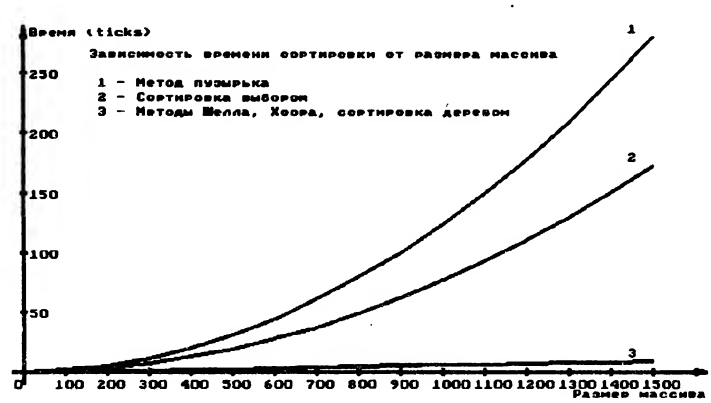


Рис. 1.

ний" элемент меньше, чем "верхний". Таким образом, мы вытолкнем наверх самый "легкий" элемент всего массива. Теперь повторим всю операцию для оставшихся неотсортированными N-1 элементов (т.е. для тех, которые лежат "ниже" первого). Как видно, алгоритм достаточно прост, но, как иногда замечают, он является непревзойденным в своей неэффективности. Немного более эффективным, но таким же наглядным является второй метод.

Сортировка выбором (листинг 2)

На этот раз при просмотре массива мы будем искать наименьший элемент, сравнивая его с первым. Если такой элемент найден, поменяем его местами с первым. Затем повторим эту операцию, но начнем не с первого элемента, а со второго. И будем продолжать подобным образом, пока не рассортируем весь массив. Авторы многих учебников по программированию (например, [3]) любят приводить в качестве примеров один из двух вышеописанных методов. С одной стороны, эти методы являются весьма наглядными, и на их примере начинающие программимсоты гут хорошо попрактиковаться в использовании циклов, но с другой стороны, оба эти метода, как уже говорилось выше, являются очень неэффективными. Можно значительно ускорить время работы по сортировке данных, если применить третий метод.

Метод Шелла (листинг 3)

Этот метод был предложен автором (Donald Lewis Shell) в 1959 г. Как отмечается в [2], основная идея алгоритма заключается в том, чтобы вначале устранить массовый беспорядок в массиве, сравнивая далеко стоящие друг от друга элементы. Как видно, интервал между сравниваемыми элементами (gap) постепенно уменьшается до единицы. Это означает, что на поздних стадиях сортировка сводится просто к перестановкам соседних элементов (если, конечно, такие перестановки являются необходимыми). Программная реализация этого метода позаимствована из работы [2]. (На взгляд автора, функция сортировки методом Шелла, предложенная в работе [2], является одной из самых лучших и универсальных. Если кто-то усомнится в этом, пусть попробует написать более совершенную). Принципиально иной подход к проблеме упорядочивания данных продемонстрирован в четвертом алгоритме.

Метод Хоора (листинг 4)

Этот метод, называемый также быстрой сортировкой (QuickSort), был разработан в 1962 г. (его разработал Charles Antony Richard Hoare). Суть метода заключается в том, чтобы найти такой элемент множества, подлежащего сортировке, который разобьет его на два подмножества: те элементы, что меньше делящего элемен-

С целью предоставления владельцам персональных компьютеров информации о фирмах, действующих на компьютерном рынке, и предлагаемых ими продуктах и услугах "Монитор" публикует на страницах журнала каталоги фирм и программ.

В каталоге фирм помещается следующая информация о фирме.

- Название фирмы, адрес, телефон, факс.
- Краткое описание продуктов и услуг, предлагаемых фирмой (не более 30 слов).

В каталоге программ помещается следующая информация о программе.

- Наименование программы.
- Как связаться с поставщиком (адрес, телефон, факс).
- Краткое описание программы и ее функций (не более 30 слов).
- Аппаратура и операционная система, необходимые для работы программы.
- Языки, на которых доступны программа и документация.
- Число пользователей в России и в других странах.
- Цена (или диапазон цен).

"Монитор" публикует также объявления фирм, желающих найти высококвалифицированных специалистов по компьютерам и программированию, и объявления частных лиц, ищущих работу в этой области.

Для получения более подробной информации Вы можете позвонить по телефону (095) 238-99-33 либо выслать открытку со своими координатами по адресу 117419, Москва, а/я 765.

Монитор-1993

Подписной купон

МОНИТОР

Адрес _____

Ф.И.О. или название _____

организации _____

В 1993 году выйдут 10 номеров "Монитора".

Подписка: перечислите 600 рублей на счет НТО "Софт-Москва": р/с 467002 в к/б "Банк развития XXI век", к/с 161816 в РКЦ ГУ ЦБ РФ по г.Москве, МФО 201791; вышлите подписной купон и квитанцию об оплате по адресу: 117419, Москва, а/я 765. Подписная цена действительна до 31.12.92

та, и те, что не меньше его. Эту идею можно реализовать многими способами. Вот один из них.

Для конечного множества элементов (массива чисел N, состоящего из 10 элементов) установим два индекса на первый (индекс i) и последний элементы (индекс j) (рис 2).

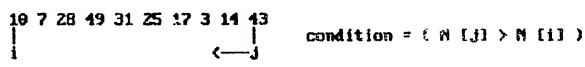


Рис. 2.

На каждом шаге будем изменять значение индекса j на значение step (вначале оно равно -1 - индекс уменьшается), т.е. двигать j по направлению к индексу i (влево). Мы будем делать так в том случае, если элемент множества с индексом j (j-й элемент) больше, чем i-й элемент. Если это условие (выраженное в переменной condition) не выполнено - поменяем местами i-й и j-й элементы множества и сами индексы i и j, а также изменим знак у значения step - оно станет равным +1 (рис. 3).

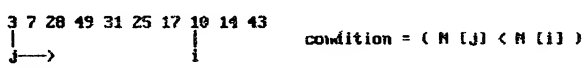


Рис.3

Продолжим процесс: индекс j опять движется к i (вправо), но теперь j увеличивается. Изменится и условие condition - сейчас для продолжения процесса необходимо, чтобы j-й элемент был меньше i-го.

Будем продолжать вышеописанный процесс до тех пор, пока индексы i и j не станут одинаковыми. В этот момент можно утверждать, что i-й (он же и j-й) элемент разделил исходное множество на два подмножества: все элементы, находящиеся слева от делящего элемента, меньшие его, и все элементы, находящиеся справа, не меньшие делящего (i-го) элемента. Получилось, что i-й элемент стоит на своем месте (рис. 4).

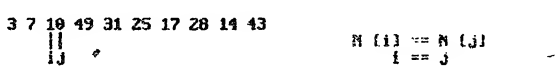


Рис.4.

Таким образом, мы описали процедуру нахождения расположения одного элемента. Иными словами, мы "отсортировали" один элемент множества. Такая же процедура применима к элементам "левого" и "правого" подмножеств, которые на данный момент еще не отсортированы. Мы описали рекурсивную процедуру, сортирующую элементы множества, начиная с элемента с индексом left и завершая на элементе с индексом right. Условие завершения процедуры - совпадение границ, которое эквивалентно тому, что в множестве остался один элемент.

В некоторых случаях (это связано с особым представлением данных) удобно использовать пятый метод .

Сортировка с помощью бинарного дерева (листинг 5)

Бинарным (двоичным) деревом назовем упорядоченную структуру данных, в которой каждому элементу (предшественнику или корню (под)дерева) поставлены в соответствие по крайней мере два других элемента (преемника). Причем для каждого предшественника выполнено следующее правило: левый преемник всегда меньше, а правый преемник всегда больше или равен предшественнику. Если мы составим такое дерево из букв слова "СОРТИРОВКА", сравнивая ASCII коды букв, то получим следующую структуру (рис. 5).

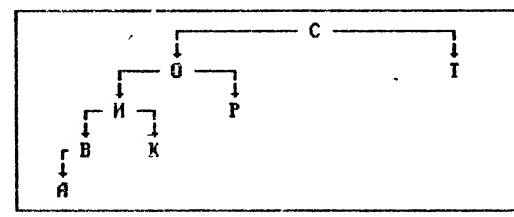


Рис. 5.

Как видно, узел "С" (узлом называют элемент дерева) имеет два преемника: левый "О" и правый "Т". Если составить бинарное дерево из элементов неупорядоченного массива, то в общем случае дерево получится достаточно хорошо заполненным (заметим, что если массив уже был рассортирован, то дерево выродится в линейный список). Если мы будем обходить дерево по правилу "ЛЕВЫЙ преемник - КОРЕНЬ - ПРАВЫЙ преемник", то, записывая все встречающиеся элементы в массив, мы получим упорядоченное в порядке возрастания множество. На этом и основана идея сортировки деревом. Этот метод имеет некоторые минусы: требуется много памяти (в зависимости от размера массива), в больших моделях памяти, т.е. при использовании 4-байтовых адресов, время работы несколько увеличивается. Но вместе с тем этот метод достаточно эффективен - по быстродействию он совсем незначительно уступает методу Шелла.

Для описания бинарного дерева используем следующую структуру:

```
typedef struct tree
{
  int a;              /* информационное поле */
  struct tree *left;  /* левый преемник */
  struct tree *right; /* правый преемник */
} TREE;
```

Тип TREE также будут иметь две вспомогательные рекурсивные функции, использующиеся в этом методе. Функция search_tree() строит бинарное дерево, добавляя в него новый элемент. Функция tree_array() используется для преобразования бинарного дерева в отсортированный массив.

В заключение хочется сделать несколько практических замечаний. Конечно не следует использовать в больших серьезных программах (особенно в математических и статистических) первые два алгоритма. Они могут быть полезны при обучении, т.к. очень просто программируются. Наиболее универсальным является, пожалуй, третий алгоритм - метод Шелла. Он также просто программируется и не требует большого количества памяти. Как отмечается в [1], он особенно эффективен при относительно небольшом количестве сортируемых элементов (скажем, N <= 1000). Другое дело, четвертый алгоритм - метод Хоора. При его использовании особое внимание следует уделять методу выбора разделяющего элемента, т.к. в некоторых случаях при неправильном выборе время сортировки может увеличиться в несколько раз (подробнее см. [1]). Он также требует большего количества памяти, т.к. используется рекурсия. Последний, пятый метод - сортировку деревом, следует применять лишь в тех случаях, когда данные уже представлены в виде, удобном для сортировки, т.е. представляют собой нечто похожее на структуру TREE, описанную выше.

Несколько необходимых замечаний к листингам

Во всех предлагаемых функциях сортировки используется макроподстановка swap() для обмена значениями двух целочисленных переменных. Этот макрос можно описать следующим образом:

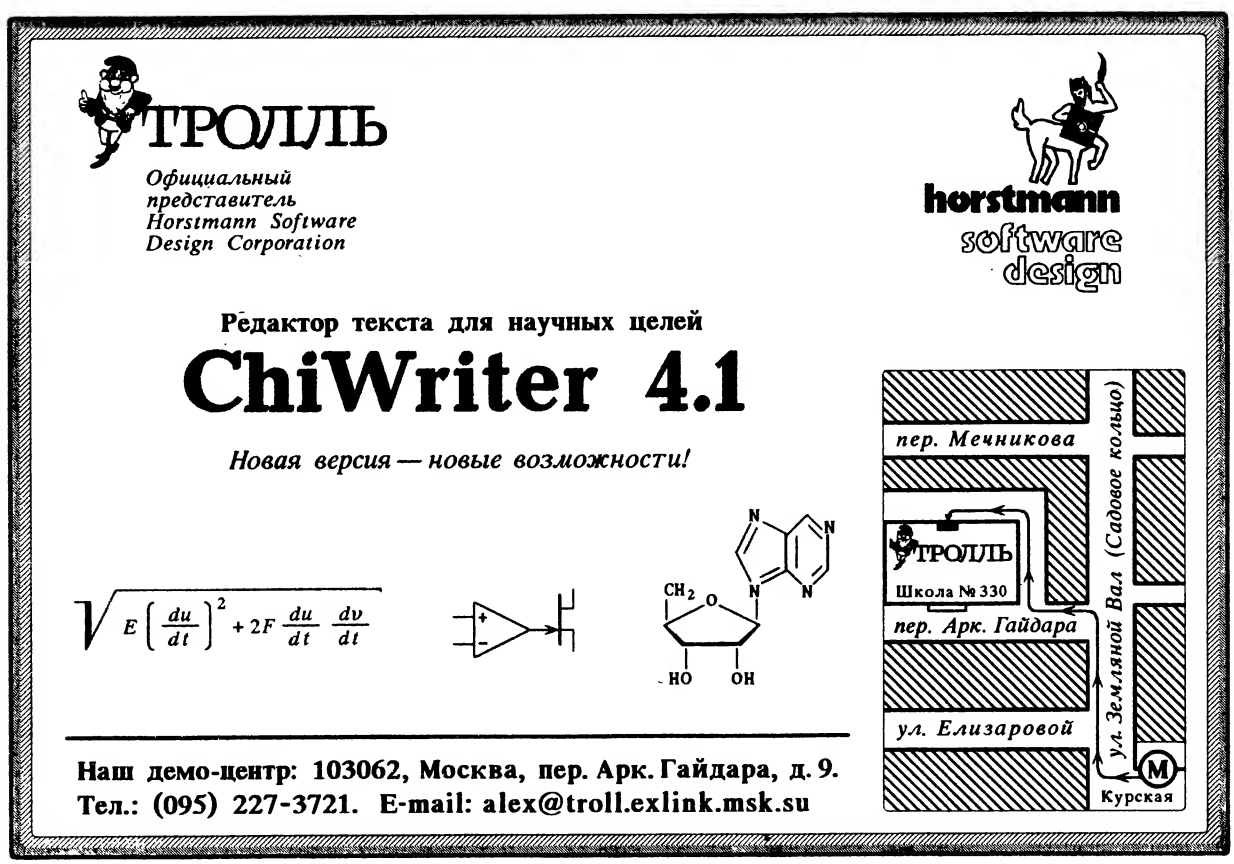
```
#define swap(a,b) { int tmp; tmp=a; a=b; b=tmp; }
```

Во всех листингах, если не оговорено иначе, целочисленный массив, подлежащий сортировке, обозначается int a[ ], размерность этого массива обозначается int dim.

Любой алгоритм сортировки (кроме, пожалуй, пятого) не должен зависеть от типа сортируемых данных. В приведенных листингах исключительно в целях упрощения был выбран целочисленный тип (int). Поскольку не имеет смысла каждый раз писать новую функцию для сортировки нового типа данных или другого условия сортировки (вместо сортировки по убыванию может понадобиться применить сортировку по возрастанию), целесообразно сделать функцию сортировки универсальной, т.е. передавать ей в качестве параметров адрес функции сравнения, специфичной для данного типа. Поскольку разные типы данных имеют разные размеры, следует также сообщать функции размер элемента сортируемого множества и количество этих элементов. ■

Литература.

1. Д. Кнут. Искусство программирования для ЭВМ. Т. 3, Сортировка и поиск. - М.:, Мир, 1978.
2. Б. Керниган, Д. Ритчи. Язык программирования Си. М.:, Финансы и статистика, 1992.
3. М. Уэйт, С. Прата, Д. Мартин. Язык Си (руководство для начинающих). - М.:, Мир, 1988.



Листинг 1. Функция сортировки методом пузырька

```
/*****************************************/
/*          Сортировка простым обменом          */
/*              (методом пузырька)              */
/*****************************************/
void sort_change(a, dim)
int a[ ], dim;
{
    int i, j;

    /* Проход массива "вниз", начиная с нулевого элемента */
    for (i = 0; i < dim; i++)
/* Проход массива "вверх", начиная с последнего до i-го элемента */
        for (j = dim-1; j > i; j--)
    /* Сравнение двух соседних элементов и их обмен */
            if(a[j-1] > a[j]) swap(a[j-1], a[j]);
}
```

Листинг 2. Функция сортировки выбором наименьшего элемента

```
/*****************************************/
/*        Сортировка простым выбором          */
/*****************************************/
void sort_choose(a, dim)
int a[ ], dim;
{
    int i, j, k;

    /* Проход массива, начиная с 0-го до предпоследнего элемента */
    for (i = 0; i < dim-1; i++)
    {
    /* Проход массива, начиная с (i+1)-го до последнего элемента */
        for (k = i, j = i+1; j < dim; j++)
            if(a[j] < a[k]) k = j; /* Поиск наименьшего k-го эл-та */
        swap(a[k], a[i]);    /* Перемещение наименьшего "вверх" */
    }
}
```

Листинг 3. Функция сортировки методом Шелла

```
/*****************************************/
/*          Сортировка методом Шелла          */
/*****************************************/
void sort_shell(a, dim)
int a[ ], dim;
{
    int i, j, gap;

    for (gap = dim/2; gap > 0; gap/=2)        /* Выбор интервала */
        for (i = gap; i < dim; i++)           /* Проход массива */
    /* Сравнение пар, отстоящих на gap друг от друга */
            for (j = i-gap; j >= 0 && a[j] > a[j+gap]; j -= gap)
                swap(a[j], a[j+gap]);
}
```

Листинг 4. Функция сортировки методом Хоора

```
/*****************************************/
/*          Сортировка методом Хоора          */
/*          (метод быстрой сортировки)        */
/*****************************************/
void sort_hoor(a, left, right)
int a[ ], left, right;
{
    int i = left, j = right, step = -1, condition = 1;
    if(left >= right) return;        /* Сортировать нечего */
    /* Сортируем с левой границы до правой */
    do {
        /* Если выполняется условие перестановки элементов ... */
        if(condition == (a[i] > a[j]))
        {
            swap(a[i], a[j]);          /* Перестановка чисел */
            swap(i, j);                /* Обмен местами индексов */
            step *= -1;                /* Теперь - в другую сторону */
        /* Обмен условия на противоположное */
            condition = !condition;
        }
        j += step;              /* Передвинем индекс */
    } while(j != i);            /* До тех пор пока индексы не сойдутся */
    sort_hoor(a, left, i-1);        /* Отсортируем то, что слева ... */
    sort_hoor(a, i+1, right);       /* ... и то, что справа */
}
```

Листинг 5. Функция сортировки с помощью бинарного дерева

```
/*****************************************/
/*          Процедура обхода дерева          */
/*          и добавления элемента            */
/*****************************************/
TREE *search_tree(root, new_value)
TREE *root;                  /* Корень (под)дерева */
int new_value;               /* Новый элемент */
{
    if (root == NULL)        /* Если дошли до корня ... */
    {                        /* ... создаем новый элемент ... */
        root = (TREE*)malloc(sizeof(TREE));
        root->a = new_value;
        root->left = root->right = NULL;
        return root;    /* ... и выходим */
    }
    if (new_value >= root->a)  /* Пойдем к корню */
            root->right = search_tree(root->right, new_value);
        else root->left = search_tree(root->left, new_value);
    return root;
}

/*****************************************/
/*        Процедура заполнения массива        */
/*****************************************/
void tree_array(root, a)
TREE *root;          /* Корень (под)дерева */
int a[ ];            /* Массив */
{
    static i = 0;          /* Счетчик элементов массива */

    if (root == NULL) return;    /* Условие окончания - */
                                 /* - найден корень */
    tree_array(root->left, a);   /* Обход левого поддерева */
    a[i++] = root->a;            /* Присвоение значения элементу */
                                 /* массива и увеличение счетчика */
    tree_array(root->right, a);  /* Обход правого поддерева */
    free(root);
}

/*****************************************/
/*          Сортировка с помощью             */
/*          бинарного дерева                 */
/*****************************************/
void sort_tree(a, dim)
int a[ ], dim;
{
    int i;
    TREE *root = NULL;

    for (i = 0; i < dim; i++)                /* Проход массива */
        root = search_tree(root, a[i]);      /* и заполнение дерева */
    tree_array(root, a);                     /* Заполнение массива */
}
```

Aho + Weinberger + Kernighan = AWK

К. Злобин

Если вы изучали язык Си по K&R (книга Б.Кернигана и Д.Ритчи "Язык программирования Си"), то, должно быть, помните, что одним из самых первых примеров (почти сразу после "hello, world") там идет программа подсчета числа слов во входном потоке (не "машинных слов", а "цепочек литер, не содержащих в себе символов пробела, табуляции и новой строки"). Впрочем, если вы ни разу не держали в руках K&R, или даже если вы презираете Си и пишете на Паскале или экзотической пока еще Модуле-2, вы согласитесь, что задачка эта типично учебная, программа на вашем любимом языке займет 10-15 строк и вообще делать тут нечего.

А теперь представьте, что Вам (давно уже опытному программисту) вдруг действительно потребовалось узнать число слов в данном конкретном текстовом файле. Что делать? Ах да, учебная задачка, 10-15 строк и вообще... А сколько времени у вас реально займет их написание - от осознания проблемы до получения результата? (Если у вас машина под рукой, попробуйте проделать это сейчас, не забудьте взглянуть на часы). Четверть часа, не меньше, из них половина на исправление глупейшей ошибки - и как вы ее ляпнули? Не жалко времени - на такой-то пустяк? Вот если бы не 10-15 строк, а две...

```
        { count + = NF }
END     { print count }
```

Устраивает? Это программа на языке AWK. Он специально предназначен для решения задач по обработке текстов. В таких задачах обычно считается, что текст состоит из строк (которые нужно обработать в цикле), а строки - из слов. И эту модель предоставляет нам язык. Программа на AWK состоит из пар вида:

```
условие1 { действие1 }
   ...
условиеN { действиеN }
```

Интерпретатор языка читает входной текст строку за строкой, для каждой проверяет по очереди все условия и в случае их истинности выполняет соответствующие действия. Посмотрим на наш пример.

Первая строка: условие опущено - по умолчанию "для всех строк". Действие - увеличить переменную count на значение встроенной переменной NF (Number of Fields), куда интерпретатор записал число слов в текущей строке.

Вторая строка: END - это такое специальное условие, которое становится истинным после последней строки текста. Действие - напечатать значение count, где у нас накопилось общее число слов в тексте.

Не правда ли, просто? Язык сделал за нас всю рутинную работу - организовал цикл по строкам, разбил строки на слова... Конечно, адепты надежного, структурного и т.п. программирования придут в ужас - мы не только не описали переменные, мы даже не дали им начальных значений! Но все эти правила дисциплины созданы для больших программ. А для таких, как наша, лучший критерий надежности - очевидность. Когда вся программа перед глазами и ее логика прозрачна, формальностями можно и пренебречь ради краткости и простоты.

В нашем примере оба условия оказались, по существу, вырожденными. А какими они еще бывают? Прежде всего, обычные логические выражения. Например, программа

```
NF > 10
```

напечатает строки входного текста, содержащие более 10 слов. Это не оговорка - вы видите законченную программу. Печать текущей строки - это действие по умолчанию.

Но чаще всего в качестве условий используются образцы. Что такое образец? В простейшем случае - цепочка литер. Например, программа

```
/Штирлиц/
```

напечатает те строки входного текста, в которых упомянут Штирлиц. Более сложные образцы - это регулярные выражения. Что такое регулярное выражение? Если вы работаете с редактором MultiEdit или пользуетесь утилитой GREP, поставляемой со всеми компиляторами фирмы Borland, они вам уже знакомы, ну а если нет - посмотрите на рис. 1.

А теперь давайте рассмотрим пример посложнее. Пусть мы хотим подсчитать не общее количество слов, а число вхождений в текст каждого слова в отдельности.

Причем словами будет считаться не "нечто, разделенное пробелами", а последовательность русских букв. По существу, мы хотим составить частотный словарь для входного текста. Программа решения этой задачи приведена на рис. 2, а здесь мы рассмотрим ее строка за строкой.

```
/Штирлиц|Исаев|Тихонов/   - годится любая из трех фамилий
/(Ш|ш)тирлиц|(И|и)саев|(Т|т)ихонов/ - можно с маленькой буквы
/0|1|2|3|4|5|6|7|8|9/     - любая цифра
/[0123456789]/            - то же, но короче
/[0-9]/                   - еще короче
/[^0-9]/                  - любой символ, кроме цифры
/М.ллер/                  - Мюллер, Миллер и даже М'ллер
/Мюлл?ер/                 - Мюллер либо Мюлер
/Мюл+ер/                  - Мюлер, Мюллер, Мюлллер...
/Мюл*ер/                  - те же плюс Мюер
/^Штирлиц/                - Штирлиц в начале строчки
/Штирлиц$/                - Штирлиц в конце строчки
/^$/                      - пустая строчка
/\.\[\\/                  - три литеры .[\
/[A-Za-z][A-Za-z0-9]*/    - идентификатор
/[+-]?[0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+)?/   - число
```

Рис. 1. Регулярные выражения

Строка 1: Условие BEGIN похоже на END - действие выполняется до считывания первой строки текста.

Строка 2: Во встроенную переменную FS (Field Separator) устанавливаем образец разделителя, по которому строки входного текста будут разбиваться на слова. В данном случае это все, кроме русских букв.

Строка 4: Пустое условие - для всех строк.

Строка 5: Заголовок цикла по словам - совсем как в Си.

Строка 6: $i - это i-е слово. A table - это ассоциативная таблица. В описании языка их назвали массивами (array), но они, во-первых, не имеют фиксированного размера, а во-вторых, вместо целочисленных индексов используют строковые ключи. Этот оператор увеличивает на единицу элемент таблицы table, соответствующий ключу $i.

Строка 8: После просмотра всего текста...

Строка 9: Удаляем из таблицы счетчик пустых слов - из-за особенностей алгоритма разбиения они могли появиться в начале и конце некоторых входных строк.

Строка 10: Цикл по всем ключам таблицы table.

Строка 11: Для каждого печатаем сам ключ и счетчик.

Ну как, понравилось? Конечно, в две строчки мы на этот раз не уместились, но сколько займет такая программа на Си или Паскале? Наличие предопределенного класса Collection в вашей реализации C++ облегчает дело, но стоит ли стрелять из пушек по воробьям?

Так что же такое язык AWK и откуда он взялся? Если вы уже посмотрели в англо-русский словарь, то, что вы там нашли (awkward - неуклюжий), к делу отношения не имеет. AWK - это просто инициалы разработчиков языка: Ахо, Вайнбергера и Кернигана (Aho, Weinberger, Kernighan). Первоначально он был реализован в среде UNIX и является там стандартным, но в последнее время появились реализации

```
BEGIN {                          # 1
   FS = "[^А-пр-я]+"             # 2
}                                # 3
{                                # 4
   for (i = 1; i < = NF; i++)    # 5
      table[$i]++                # 6
}                                # 7
END {                            # 8
   delete table[""]              # 9
      for (w in table)           # 10
      print table[w], w          # 11
}                                # 12
```

Рис. 2. Частотный словарь

и для MS-DOS. Лучшей из них, по-видимому, является MAWK Майка Бреннана (Mike Brennan), которая (что отрадно в нашу эпоху коммерциализации всего и вся) распространяется свободно в рамках проекта GNU (текст GNU GENERAL PUBLIC LICENSE, входящий в комплект поставки, сам по себе весьма интересен). Объем самого интерпретатора - 55K, описания языка и реализации - 13K. Более подробное описание языка содержится в книге его разработчиков: Aho, Kernighan and Weinberger, "The AWK Programming Language", Addison-Wesley Publishing, 1988. К сожалению, на русский язык она не переведена. Быть может, эта статья пробудит интерес к языку программирования, хотя и не претендующего на вытеснение всех прочих, но очень удобного в своей области применения. Попробуйте поработать с AWK - не пожалеете! ■

Работаем с таблицами

М. Черкашин

Предположим, Вам хочется написать небольшой калькулятор, то есть программу, которая способа читать выражение с клавиатуры, вычислять значение выражения и печатать его. Вы можете действовать по следующему алгоритму.

Заметим, что у данного алгоритма есть одна особенность. Какими бы ни были операции и приоритеты операций и сколько бы их ни было, алгоритм остается неизменным. Если бы мы пытались реализовать этот алгоритм вручную и не помнили всех операций, мы бы выписали обозначения всех операций и их приоритеты в таблицу и пользовались бы ею по мере необходимости.

Когда те же данные хранятся в памяти машины, также говорят, что такая-то информация хранится в таблице. Можно сказать так: структуры, информация в которых не изменяется в течение работы программы, называются таблицами.

Программа для этого алгоритма приведена в примере 1.

Сюрприз

Таблицы хранят в себе много сюрпризов для программиста - как приятных, так и неприятных. А вот и один из них. Мы можем записать в таблицах программу, решающую нашу задачу; а программой, интерпретирующей таблицы, будет транслятор. Почему бы и нет? Мы можем даже рассматривать любую нашу программу как таблицы для компьютера.

Конечно никто не записывает в таблицы программы на Паскале. Но наш пример говорит о том, что таблицы - это невероятно мощная штука. Вот, например, неслыханный прогресс в искусственном интеллекте, связанный с появлением экспертных систем, связан в основном с тем фактом, что удалось поместить в таблицы (а не в текст программы) почти всю информацию, определяющую функционирование системы.

Форма Бэкуса – Наура

Для того чтобы таблицы было можно использовать, необходимо выбрать форму представления синтаксиса программы, одинаково понятную как человеку, так и машине. Существуют несколько таких форм представления. Здесь выбрана одна из них - самая, по мнению автора, удобная.

Любая программа состоит из конструкций - фрагментов, которые имеют какой-то смысл: операторов, описаний и т.д. Более того обычно бывает, что конструкция каждого из возможных типов - это строго заданная последовательность конструкций других типов, либо другая подобная последовательность и т.д.

Пример использования формы Бэкуса-Наура приведен на рис. 1.

Символ : означает "определяется как". Обратим внимание, что конструкция <Хвост> - последовательность операторов, заканчивающихся ключевым словом end - определяется через самое себя: <Хвост> это либо ключевое слово end, либо оператор, за которым следует <Хвост>. Но это не должно нас удивлять. Вы же знаете, как образуется очередь? Сначала стоит один человек, который уже в каком-то смысле очередь. Потом к нему подходит второй - и они уже вдвоем образуют очередь, потом еще один. Так и получается, что любая очередь состоит из последнего в этой очереди и очереди, которая стоит перед ним.

Nota bene

Процедура Вычислить. Достать из стека значений два значения, из стека операций - одну операцию. Применить операцию к значениям и полученный результат поместить в стек операций.

Конец процедуры Вычислить.

Алгоритм вычисления значения выражения. Поместить в стек операций Прокладку. Пока не встретится конец выражения, считывать лексемы и в зависимости от типа лексемы производить следующие действия.

1. Если считанная лексема есть число, положить его в стек значений.
2. Если считанная лексема есть операция, положить ее в стек операций.

Однако класть операцию с более низким приоритетом на операцию с более высоким приоритетом нельзя. Поэтому в случае если у считанной операции приоритет ниже, чем у лежащей в стеке наверху, применить процедуру Вычислить столько раз, сколько необходимо.

3. Если считанная лексема есть открывающая скобка, то положить в стек операций Прокладку.
4. Если считанная лексема есть закрывающая скобка, то применять процедуру Вычислить, пока на вершине стека операций не появится Прокладка. Когда встретится конец выражения, применять процедуру вычислить, пока не появится Прокладка. Единственное оставшееся в стеке значений число и будет искомым значением выражения.

Конец алгоритма.

Синтаксически управляемый перевод

С нашей точки зрения любой компилятор осуществляет перевод - с языка, который воспринимается компилятором, на язык, программу на котором мы можем обработать. Это может быть Паскаль (если у нас изначально есть с него компилятор), это может быть язык ассемблера, это может быть машинный язык и т.д.

Однако попробуйте описать этот перевод - и у Вас возникнут определенные проблемы. Но если мы вспомним формулы Бэ-

Пример 1. Маленький калькулятор с таблицей

```
procedure OneOperation;
var X,Y,Z : integer;
   op : char;
begin
   X := PopVal;
   Y := PopVal;
   op := PopOper;
   case op of
         '+': Z := X+Y;
         '-': Z := X-Y;
         '*': Z := X*Y;
         '/': Z := X div Y;
   end;
   PushVal (Z);
end;

Procedure Compute;
begin
   PushOper (Layer);
   case LexMode of
         LexNumber:   PushVal (LexVal);
         LexOper:     begin
                           while Priority (TopOper)
                                 do Compute;
                           PushOper (Chr (LexVal));
                      end;
         LexLeft:     PushOper (Layer);      (*(*)
         LexRight:    begin
                           while TopOper < > Layer
                                 do Compute;
                           PopOper;
                      end;
   end;
end;

begin
   (* Scan - это сканер *)

   while not Eof (T) do begin
         Scan;
         Compute;
   end;
   (* А теперь берем в стеке значений то, что осталось - *)
   (*                  и это и есть результат              *)
end.
```

куса - Наура, все совершенно изменится. В самом деле, они задают нам структуру программы.. А это совершенно меняет дело. В этом случае мы уже можем сказать: <Оператор> транслируется в то-то и то-то. Например, если исходная программа есть

<Описания>
begin
<Хвост>

(а так может выглядеть программа на Паскале), то мы должны получить результат - его отныне для краткости будем называть "кодом":

скомпилированные <Описания>
main ()
{
скомпилированный <Хвост>

Таким образом, из программы на Паскале мы получаем программу на Си.

Приведенный только что пример показывает, как конструкция одного языка может быть переведена в конструкцию другого. В самом деле, представим его в виде следующей схемы:

Было:	Стало:
<Программа>:	<Программа>:
<Описания>	<Описания>
begin	main () {
<Хвост>.	<Хвост>

С помощью формул Бэкуса - Наура можно разложить любую программу на конструкции, их - на составляющие их конструкции и т.д. Если к каждой формуле Бэкуса - Наура мы добавим формулу из части "Стало", то мы полностью зададим таким образом перевод из исходного языка в "код". Такой подход является вполне естественным: мы как бы разбираем программу на части, а потом снова собираем, но уже чуть-чуть по-другому.

Такой набор пар правил называется схемой синтаксически управляемого перевода(СУ-перевод). В качестве примера можно привести схему для перевода из подмножеств языка Паскаль в подмножество языка Си (рис. 1).

Было:	Стало:
<Программа>: program <Описания> <Тело>	<Программа>: <Описания> <Тело>
<Описания>: var <Переменная> ,<ХвостПеременных>;	<Описания>: int <Переменная> ,<ХвостПеременных>;
<ХвостПеременных>: :integer;	<ХвостПеременных>: ;
<ХвостПеременных>: <Переменная> ,<ХвостПеременных>	<ХвостПеременных>: <Переменная> ,<ХвостПеременных>
<Тело>: begin <ХвостОператоров>.	<Тело>: main (){ <ХвостОператоров>
<ХвостОператоров>: <ХвостОператоров> <Оператор>	<ХвостОператоров>: <ХвостОператоров> <Оператор>
<ХвостОператоров>: end	<ХвостОператоров>: }
<Оператор>: <Переменная>:= <Выражение>	<Оператор>: <Переменная>:= <Выражение>;
<Выражение>: <Число>	<Выражение>: <Число>
<Выражение>: <Переменная> <Операция> <Переменная>	<Выражение>: <Переменная> <Операция> <Переменная>
<Операция>:+	<Операция>:+
<Операция>:-	<Операция>:-
<Операция>:*	<Операция>:*
<Операция>:div	<Операция>:/
<Оператор>: if <Переменная> =<Переменная> then <Оператор> else <Оператор>	<Оператор>: if (<Переменная> ==<Переменная>) <Оператор> else <Оператор>
<Оператор>: while <Переменная> =<Переменная> do <Оператор>	<Оператор>: while (<Переменная> ==<Переменная>) do <Оператор>
<Оператор>: begin <ХвостОператоров>	<Оператор>: {<ХвостОператоров>
<Оператор>: writeln (<Переменная>);	<Оператор>: printf ("%d", <Переменная>);

Рис. 1. Схема синтаксически управляемого перевода

А в виде программы?

Схемы синтаксически управляемого перевода были бы ни на что не годны, если бы не тот факт, что их можно интерпретировать совершенно механически - то есть с помощью программы. А раз так, то можно написать программу интерпретации схемы перевода, поместить сами схемы в таблицы (т.е. в оперативную память) - а далее запускай программу и иди обедай.

В свете того, что сейчас было сказано, самый естественный алгоритм СУ перевода выглядит так.

1. Разобраться в структуре входного текста.
2. Для каждой конструкции произвести замену в соответствии с нашей схемой.

Иными словами, разбить программу на операторы и каждый оператор транслировать. К сожалению, буквально этот способ как правило оказывается неудовлетворительным. Поэтому не будем пытаться справиться совсем уж со всеми схемами СУ переводов, а наложим на них два ограничения. Эти ограничения не очень-то нас стеснят, зато сильно упростят написание программы.

Во-первых, разобраться в структуре входного текста не всегда так легко. Мы предполагаем, что во входном языке по первой лексеме конструкции можно определить, что это за конструкция. Так оно обычно и бывает. Тип оператора (условный или цикла или какой-нибудь еще) определяется первым ключевым словом. Описние процедуры и функции также отличается по первому ключевому слову. И так далее.

Во-вторых, мы не можем позволить себе роскошь держать в памяти весь текст программы целиком. Поэтому в одном правиле в частях "Было" и "Стало" названия конструкций должны идти в одном и том же порядке - такое наше второе ограничение. Стало быть подобные пары правил просто запрещены:

Было:	Стало:
<Программа>:	<Программа>:
<Описания>	begin
begin	<Хвост>
<Хвост>.	where
	<Описания>

И понятно, для чего нам это нужно. Вот считываем мы исходную программу, разбираемся с описаниями и уже готовы переписать их скомпилированными в выходной файл - с глаз долой, из сердца вон - как выясняется, что сначала мы должны записать скомпилированный <Хвост>, а описания должны пока держать не пойми где - то ли в оперативной памяти, то ли на диске. Очень неудобно получается.

А теперь посмотрим программу. Основная ее часть - это процедура Разбор. Она осуществляет перевод конструкции, название которой ей дано в качестве параметра. Например, если в качестве параметра ей дано название <Оператор>, то она должна считать из входного файла один оператор и записать в выходной файл соответствующий ему "код". А если в этот оператор в качестве составных частей входят другие конструкции, то эта процедура вызывает самоё себя (рекурсивно) для перевода этих конструкций. И для того чтобы запустить перевод, достаточно вызвать процедуру Разбор с параметром <Программа> - и все!

Таблиц в этой программе две. Первая - основная - содержит сами правила СУ-перевода (см. рис. 1). Вторая для каждой конструкции и каждого типа лексемы содержит правило.

И еще одно. Мы до сих пор так и не упомянули о том, с какого языка и на какой мы транслируем. Но в том-то и дело, что это зависит только от таблиц. В примере мы осуществляем перевод с небольшого подножества языка Паскаль в подмножество языка Си, но ведь стоит изменить таблицы, мы уже получим нечто другое: то, что нам нужно. Программа (пример 2).

Работа над ошибками

Вообще существуют два основных принципа написания компиляторов. Придумать третий, по всей видимости, невозможно. Либо "по одной процедуре на каждую конструкцию" - либо "задаем синтаксис в одном месте (например в таблицах), а сам компилятор - в другом". Используя первый принцип, часто приходится отдельно ловить ошибки в каждом операторе. А это малость муторно. А то есть такое правило работы водолаза: "Если встретил акулу - по морде ее!" Однако инструкцию типа: "Закрутил гайку, посмотри, если есть акула - по морде ее; положил гаечный к инструментам, а если рядом акула..." ни один водолаз читать не будет. А если программа написана таким образом, то ее ни один программист не сможет отладить.

На самом деле "ловить" ошибки не сложно. Достаточно лишь действовать по принципу: "Если что-то идет не так - ругайся!" И так поступают все компиляторы - именно поэтому так многочисленны ошибки типа "должно быть двоеточие, оператор, ключевое слово do и т.д."

Пример 2. Программа синтаксически управляемого перевода

```
type sline = string [32];
procedure Parse (NonTerminal: sline);
var i : integer;
begin

        .....
        (*
            Здесь мы определяем, какое из правил для данного
                нетерминала мы используем:
                WasRule   - Правило "Было"
                BecomeRule - Правило "Стало"
        *)

while i < = length (WasRule) and Not Error do
  begin
      Ch := WasRule [i];
      if Ch = '<' then begin
      (* Нетерминал *)
      s := '';
      Inc (i);
      while WasRule [i] < > '>' do begin
          s := s + WasRule [i];
          Inc (i);
      end;
      while BecomeRule [k] < > '<' do begin
          Write (OutFile, BecomeRule [k]);
          Inc (k);
      end;
      Inc (k);
      t := '';
      while BecomeRule [k] < > '>' do begin
          t := t + BecomeRule [k];
          Inc (k);
      end;
      Inc (k);
      if s < > t then begin
          WriteLn ('Ошибка в системе правил');
      end;
      if s = 'Переменная' then begin
          t := '';
          while Ch in ['a'..'z','A'..'Z','0'..'9'] do begin
              t := t + GetNextChar;
          end;
          Write (OutFile, t);
      end else if s = 'Число' then begin
          t := '';
          while Ch in ['0'..'9'] do begin
              t := t + GetNextChar;
          end;
          Write (OutFile, t);
      end else begin
          Parse (t);
  end;
end;
```

Проблема в ином. Если "не то" все-таки произошло, то нужно проскочить опасное место и сделать вид, что все в порядке. Это процедура называется восстановлением. Например, можно дойти до ближайшей точки с запятой и там пытаться транслировать очередной оператор. Если у нас работает схема СУ-перевода, то все ошибки сводятся к одной: не удается применить ту или иную формулу Бэкуса - Наура. А с одной-то ошибкой и справиться легче.

И наконец...

Конечно же, это не все... Например, компиляторы часто еще и сами порождают таблицы, для того чтобы другая программа, отладчик, например, смогла разобраться в коде. Или, например, компиляторы копиляторов - программы, которые по описанию языка сами строят табличный компилятор. Историй про таблицы много - и каждая словно детектив. А эта статья - это только начало.

Литература

1. Грис Д. Проектирование компиляторов для цифровых вычислительных машин. - М.:Мир, 1975.
2. Ахо А, Ульман Дж. Теория синтаксического анализа, перевода и компиляции. - М.:Мир, 1975.
3. Вирт Н. Алгоритмы + Структуры данных = Программы. - М.:Мир, 1985.

"У CA самые серьезные намерения в России..."

В июне 1992 года самый большой в мире независимый производитель программного обеспечения фирма Computer Associates (CA) объявила о приобретении фирмы Nantucket Corporation. СП "Магнит", являвшееся филиалом фирмы Nantucket в странах СНГ и Прибалтики, стало теперь представительством фирмы CA в России. Президент представительства Александр Гольберг любезно согласился дать интервью для нашего журнала.

Монитор. *В нашей стране широко известны такие продукты фирмы CA, как Clipper и SuperCalc. Сама же фирма CA только начинает свою деятельность в России. Не могли бы Вы рассказать подробнее о CA?*

А. Гольберг. Фирма Computer Associates основана в 1976 году. Ее штаб-квартира находится в городе Исландия в штате Нью-Йорк. CA стала первым независимым производителем программного обеспечения с годовым оборотом, превышающим миллиард долларов. В этом году годовой оборот CA - более 1,5 млрд. долларов. Это дает фирме возможность выделять ежегодно более 170 миллионов долларов на создание новых и совершенствование уже существующих программных продуктов.

CA производит и поставляет более 200 программных продуктов для больших и средних машин и около двух десятков - для персональных компьютеров. С программными продуктами CA работают как сотни тысяч небольших фирм во всем мире, так и крупнейшие корпорации. Более 90% компаний, входящих в список 500 крупнейших, ежегодно публикуемый журналом Fortune (Fortune 500), используют программные продукты CA. Сама фирма CA также входит в Fortune 500.

М. *Каковы планы CA в России? Готова ли фирма CA активно работать на российском рынке и вкладывать здесь деньги?*

А.Г. У CA самые серьезные намерения в России. Во-первых, CA пришла не на пустое место. Вместе с СП "Магнит" CA приобрела созданную за два года работы инфраструктуру - более чем 70 дилеров в 50 городах бывшего СССР, 20 центров обучения, многочисленные клубы программистов. Накопленный нами за это время опыт работы на российском рынке программного обеспечения позволяет рассчитывать на успешное распространение продуктов CA.

Во-вторых, фирма CA намерена инвестировать здесь деньги, в первую очередь, в локализацию и распространение программных продуктов CA. К уже имеющимся на рынке Clipper 5.0 и Clipper Tools One в ближайшее время добавятся уже локализованные CA-Clipper Tools II и CA-SuperCalc. Ведется работа по локализации других продуктов CA.

М. *Многочисленных пользователей системы Clipper, несомненно, интересуют ее перспективы после перехода Clipper "под юрисдикцию" CA.*

А.Г. CA осознает важность и значимость Clipper для его многочисленных пользователей. Поэтому CA вкладывает значительные деньги в дальнейшее развитие Clipper. Как я уже сказал, в ближайшее время мы начнем распространение функциональной библиотеки Clipper Tools II. Мы будем продолжать активную работу с пользователями Clipper. 1 - 3 декабря 1992 года в Обнинске состоится вторая конференция разработчиков Clipper.

М. *Какие программные продукты, помимо уже упомянутого Clipper, CA предполагает распространять в России?*

А.Г. CA предлагает широкий спектр программных продуктов как для больших и средних, так и для персональных компьютеров. Пользователей программных продуктов CA для персональных компьютеров можно разбить на три группы: профессиональные разработчики программного обеспечения, дизайнеры, конечные пользователи (менеджер, бухгалтер, клерк).

О продуктах для профессиональных разработчиков мы уже говорили.

Для дизайнеров CA предлагает мощные профессиональные системы создания и обработки изображений CA-Cricket Paint и CA-Cricket Image.

Особое внимание читателей "Монитора" я хотел бы обратить на три продукта CA, предназначенных для конечного пользователя.

Первый из них - это пользующийся большой популярностью у нас в стране табличный процессор SuperCalc.

Второй - полностью настраиваемая бухгалтерская система ACCPAC, которая является ведущей системой управления финансами в США,

щей системой управления финансами в США, Канаде и многих других странах. Например, в Англии на долю ACCPAC приходится около 90% бухгалтерских систем. С ACCPAC ежедневно работают десятки тысяч организаций, от маленьких частных компаний до крупных корпораций. Работая с ACCPAC, Вы получаете полную коммерческую информацию, реальную, легко доступную, профессионально представленную.

Третий продукт предназначен для менеджера - это средство управления проектом CA-SuperProject. SuperProject использует испытанные методы управления проектом, с точки зрения таких критериев, как время, стоимость, цели, ресурсы и является прекрасным помощником для любого руководителя.

В настоящее время уже локализован SuperCalc, и мы ведем работу по локализации ACCPAC и SuperProject, так что и эти два продукта скоро станут доступны нашим пользователям.

М. *Существует ли некая общая идеология или среда работы, объединяющая выпускаемые CA продукты?*

А.Г. Да, это, несомненно, так. Существует преемственность пользовательского интерфейса; это означает, что если у Вас уже есть опыт работы с программными продуктами CA, то, начиная работать с новым продуктом, Вы попадаете в уже знакомую Вам среду. Кроме того, программные продукты CA совместимы по форматам данных.

М. *Насколько мне известно, журнал "Nantucket News" - единственный выходящий на русском языке специализированный, то есть посвященный работе с одним конкретным программным продуктом, журнал для программистов. Какова его дальнейшая судьба?*

А.Г. "Nantucket News" будет преобразован в журнал "Computer Associates News". "Computer Associates News" будет по-прежнему много внимания уделять системе Clipper, однако тематика его станет гораздо шире и охватит весь спектр продуктов, предлагаемых фирмой CA. С января 1993 года журнал будет распространяться по подписке.

М. *Что еще хотели бы Вы сказать нашим читателям?*

А.Г. В заключение я хотел бы подчеркнуть, что, приобретая программные продукты фирмы CA, Вы приобретаете не просто высококачественные программные продукты. Вы начинаете работать с теми программными продуктами, которые используют в своей повседневной деятельности крупнейшие фирмы по всему миру. Программные продукты CA дают Вам полный набор необходимых для эффективной деятельности фирмы программных средств, а также позволяют организовать Вашу работу в соответствии с мировыми стандартами.

М. *Благодарю Вас за интересное интервью.* ■

Адрес представительства CA в России: 127018, Москва, 2-я Ямская, 15.
Тел. (095)289-44-77
Факс (095)289-34-69

Интервью брал Дмитрий Бройтман

▸ *стр. 19, 24*

Следует иметь в виду, что русификаторы типа BETA в этом режиме работать не будут. Это означает, что русский текст вы сможете выводить только при аппаратной реализации кириллицы или при полной загрузке русских шрифтов в принтерах серии FX.

Конечно, это всего лишь одна из тех многочисленных возможностей, которые предоставляет использование принтеров EPSON. Но, будем надеяться, ее использование будет вам полезным, а главное, она заставит открыть заброшенную документацию, полученную вместе с принтером. Кто знает, какие тайны она хранит? Читайте документацию, господа!

В.Водолазкий ■

ЛИТЕРАТУРА

1. **EPSON LX-800. 9-Pin Dot Matrix Printer. Users guide.**
2. **TechHelp 4.0 Dan Rollings, 1990.**

Листинг 1

```
  name ekonom

;
; Программа установки шрифта "лилипут"
; предназначена для экономного расхода бумаги
; на принтерах EPSON - LX,FX,MX серий.
;
        model small

        .radix 16

cr      EQU     0dh         ;для удобства ввода сообщений программы
lf      EQU     0ah

  .stack    50

.data

copyrt db   'Программа установки шрифта "лилипут". ',cr,lf
       db   '(C) B.B.Водолазкий  1992 год',cr,lf,lf
       db   '$'

LILICOD   DB  27D,'S',1,0FH,27D,'A',6

CODELEN EQU     $ - LILICOD

  .code

        MOV     AX,@DATA
        MOV     DS,AX           ;настраиваем сегмент данных
        MOV     DX,offset COPYRT
        MOV     AH,9
        INT     21              ;выводим авторскую метку
        MOV     AH,1            ;инициализируем порт принтера
        MOV     DX,0            ;обслуживаем только LPT1:
        INT     17
        CLD
        XOR     DX,DX
        MOV     CX,CODELEN
        LEA     SI,LILICOD
@@l1:
        LODSB
        MOV     AH,0
        INT     17
        LOOP    @@L1
OKStop:
        MOV     AX,4C00
        INT     21              ;Нормальный останов

  end                                                          ■
```

Журнал "Монитор" приглашает к сотрудничеству:

- Дилеров по распространению журнала
- Специалистов по анализу состояния компьютерного рынка
- Авторов оригинальных статей по программированию и вычислительной технике

Тел. (095)238-99-33
Факс (095)238-32-78
Адрес для писем: 117419, Москва, а/я 765.

Лучшая продукция и компетентность дилеров—вот что поднимает MERISEL к высотам коммерческого успеха.

Если Вы дилер высокого класса, сотрудничайте с фирмой MERISEL и тогда MERISEL предложит Вам:

- льготную систему вознаграждения
- широкую программу обучения
- содержательные семинары, форумы и выставки.

Если Вы дилер MERISEL , Вы предложите своим клиентам:

- 10 000 наименований высококачественной техники 700 ведущих фирм мира
- цены на 15—20% ниже европейских

и еще:

- комплектование индивидуальных заказов и поставку оборудования со складов в Москве
- техническое обслуживание вычислительной техники
- информационно—консультативное сопровождение программных средств
- регистрацию пользователей программных средств и поставку новых версий.

Высокий бизнес ждет Вас, господа дилеры,

а Merisel ждет Вас по адресу:

109044 Москва, Крутицкий вал, д.3, корп. 2.
Телефоны: (095) 276—9008, (095) 276—3149, (095) 276—4718.
Факс: (095) 274—0097.

MERISEL C.A.T.Ltd

Объявление

Журнал "Монитор" создает дилерскую сеть...

Информация

Своим региональным представителям "Монитор" предлагает:

- **Оптовые партии нашего журнала для распространения.**
- **Программные продукты для реализации в своем регионе с широкой системой скидок.**
- **Регулярную публикацию рекламных объявлений на льготных условиях.**
- **Оперативную информацию о состоянии рынка.**

Для получения более подробной информации вышлите по адресу: 117419, Москва, а/я 765 письмо, содержащее краткие сведения о Вашей фирме, регионе, в котором Вы работаете, какие программные продукты Вас интересуют, какое количество экземпляров журнала Вы хотели бы реализовывать.

Справки по тел. (095) 238–99–33

Факс: (095) 238–32–78.

Вместе с нами –
к стабильности и новым перспективам